UNIVERSITY OF OSLO
Department of Informatics

Investigating the Performance, Scalability & Reliability of a Distributed File System: Ceph

Master Thesis

Addisu Tesfaye Yimer

Network and System Administration

Oslo University College

May 24, 2011

Abstract

File system is the very crucial part of all computer systems. It has been improved from local to network and now to distributed file systems. NFS, SAN, and NAS are becoming obsolete as they all pursue the traditional block based storage systems and impose the whole work on a single server. The recently innovated DFSs pursue object-based storage system which helps them to decouple metadata from actual data and hence avoid single point of failure and increase performance, scalability, and reliability.

The recently added to Linux DFSs family is a POSIX compatible software called Ceph. It has developed and implemented some special concepts and unique features in its architecture (dynamic metadata subtree partitioning, intelligent object-based storage, data replication across nodes, fault tolerance, assuming nodes failure as a norm rather than exception etc). This research paper investigates and evaluates the author's claim that ceph is capable of handling more than petabyte-scale of storage in heterogeneous systems with excellent performance, high scalability and high reliability.

Acknowledgements

"For all things are from Him, by Him, and for Him; Glory belongs to Him forever! Amen." Romans 11:36

Below God my first and deepest gratitude goes to my beloved wife, Eskedar Kefialew. Tgye, you are my strength, my courage, and the love of my life. You played a great role for the success of this thesis and in my whole study from the beginning till today. Thank you so much for making my life so meaningful and joyful. Because of your lovely personality, life is so fun for me. My love to you is so much deep as always.

I'm heartily thankful to my supervisor, Ismail Hassan, for his great support, motivating discussions and encouragement. Dear Ismail, I'd like to show my gratitude to you for keeping your door open always to listen, generously support and contribute your invaluable ideas. I also would like to extend my deepest gratitude to Kyrre Begnum. Dear Kyrre, even though you didn't supervise me directly in this project, your influence has been so enormous. I acquire the very basic and key principles and advanced way of doing research and how to write a thesis from your wonderful and high standard teachings. I also can't wait to highly appreciate and be thankful to Hårek Haugerud, and Thor E Hasle. Dear Hårek and Thor, I've also learnt a lot from you both; especially Hårek, I've got so much useful ideas and principles from you in the very beginning of my study in the field. More than that both of you have been like a father to me. Because of you, I feel at home when I'm at HIO. Thor, I never forget the special care I've got from you in my challenging times. You've used your profile to help, encourage, and lift up others by all means rather than looking for a law to oppress others. You all deserve especial appreciation.

I again would like to extend my appreciation and gratitude to Evangelist Aklilu Tefera and Solomon Legesse who have been my courage and moral to keep my spirit high so as to achieve my goals. I really appreciate both of yours positive attitudes towards others and being satisfied with the success of your brothers. Dear Ake, your blessing preaching in the church, your deep understanding of the value of knowledge, your invaluable advices has great contribution in the success of this thesis too. I also thank all my classmates and many other brothers and sisters who have been supporting me in one way or the other.

Last, but not least; it is an honor for me to express my deepest gratitude to my parents for their special love to their children. Dad, you are such a wise, blessing, and wonderful father to us that from the very childhood till today I've acquired a lot of wisdom from you in many areas of life including science, of course. Mum, you are so much loving mother not only for us, but also for many others. I'm very proud to have been raised in that lovely family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

I've been very interested with the concept of Cloud Computing (CC) since I came to know about it. Getting easy, scalable, and location-independent computing resources and IT services from the cloud with paying-as-you-go [42] pricing model has impressed me so much. Thanks to the fast network (Internet) facility, and the technologies of virtualization, distributed systems, cheap storage, etc... that there is a cloud computing alternative for almost all computing services including storage with reasonable price [42]. I've also been an active user of CC. Since it satisfies the need of different IT-services requirement of small companies and private users; Since it highly saves computing resources for the world; since it is easy to use, elastic and highly scalable [42, 38]; etc..., I've been motivated to contribute something in this highly significant innovation.

To provide different IT-services for unpredictable number of customers demands the cloud providers to have highly elastic, scalable, available, and reliable systems with excellent performance. Innovation and advancement of virtualization and distributed computing are the most important technologies which help CC to exist [42, 28]. File systems are also now distributed and virtualized in order to fulfill the high need of performance, scalability and reliability of systems and in order to handle extremely large data storages and hundreds and thousands of clients.

Ceph is one of the very recently innovated distributed file systems (DFSs) which is extremely scalable, highly reliable and with excellent performance as per [1]. There are some very interesting features and special concepts developed and implemented in ceph DFS which makes it different from other DFS products [1]. Some of them are [12, 24, 11, 1]:-

- Maximum separation of metadata from actual data and independent metadata management.

- Intelligent, reliable and autonomic Object-based Storage Devices (OSDs) and Metadata Servers (MDSs)
- Dynamic sub tree management by MDSs
- Extremely scalable and excellent performance and handling storages more than Petabyte of size.
- File system work load distribution (load balance) among many servers which improves performance and scalability in a great impact.

Currently; there is high demand of big size storage systems highly scalable and reliable, with high performance file system which is capable of allowing parallel access to more than thousands of clients. In today's many datacenters most commonly used file system technologies are NFS, NAS, and SAN [11]. None of them are capable of fulfilling the above demands satisfactorily; especially they all impose the workload on a single server and use the traditional block-based storage system. This is my main motivation to work on this project. There are, of course, some numbers of distributed file systems out there other than ceph; like: Lustre, GlusterFS, pNFS, GFS, GPFS, XtreemFS, MogileFS, etc...

Best way of evaluating ceph could be by comparing its performance and scalability against all available DFSs to have a better observation of where it stands compare to other DFSs. However; since almost all DFSs are quite complex systems to setup and configure them in order to make them work and since there are time, resource, and expertise limitations, I had to choose one of them for investigation. I put some basic criteria to choose from the many DFSs. These criteria are:

- Open source,
- POSIX-compatible,
- Simplicity,
- File system and/or data size limitation,
- Incorporating latest technological concepts in its architectural approach, like:
  - Separation of metadata from actual data,
  - Object-based storage system instead of block-based,
  - Data replication to increase reliability,
  - Fault tolerance to avoid single point of failure,
  - Assuming node failure as a norm rather than exception,
  - Dynamic sub tree partitioning management,

According to the above criteria, all DFSs mentioned above fail at least in one of the criteria except ceph. In addition to the above concepts, Ceph has used some

more interesting designs in its architectural approach. It is, of course, hard to know where ceph stands in its performance, scalability, and reliability without comparing it with other DFSs. However, in addition to knowing all those architectural advantageous, investigating its performance, scalability and reliability will help to see how promising it is and then depending on the result, it could be recommended for many datacenters for their file system solutions.

1.2 File System Overview

File system is one of the most essential parts of all UNIX and Linux systems as almost everything is treated as a file (like: texts, directories, images, videos, programs, services, sockets, etc...) are all seen as files by UNIX/Linux systems. For instance, directory is seen as a file which contains lists of other files [17]. The term file system could be viewed in two ways. The most common view of file system is its directory structure or mainly referred to as directory tree which starts with forward slash / - the parent of the entire hierarchy of file system directories of Linux systems. Below the root directory ('/') there are series of subdirectories and each subdirectory may again contain other subdirectories and so on... This hierarchy of directories or directory tree is very useful to organize different kind of files according to their type, meaning and purpose in a computer system. This view point of file system can be defined as **Logical File System** [34].

File system is not only just a directory tree, it is more than that. The other view point of file system is mainly the type of software which does the job - including organizing the hierarchy of directories in addition to facilitating writing, saving and retrieving of files and/or directories from disks. The software (i.e. a file system) needs some section or partition in a hard disk to reside. These kinds of software are called **Physical File Systems** or more commonly they are called **Types of File Systems** [34]. Examples of such types of file systems are: ext2, ext3, reiserFS, btrfs, NFS, SMB, ceph, etc...

Our focus in this project is on the second meaning or view of file system which is **Types of File Systems** or **Physical File Systems** - focusing on the software which plays the main role of file systems. Since the invention of Linux in 1970's, there have been different types of innovations and developments on file systems to fulfill the need of its times. The first file system used by UNIX was called FS, and then improved to FFS (Fast File System) [34]. Minix was the first file system used by Linux, and then improved to extended file systems (i.e. ext2, ext3, ext4), [26]. There are also other different types of Linux file systems as mentioned above. In general we can again group the **Types of File Systems** in to three [11]:

i **Local File System**

- The whole file system resides in local machine.

- Access thru network from remote machine is not possible.
- Examples: FFS, ext2, ext3, ext4, reiserFS, DualFS, btrfs, etc...

ii **Network File System**

- Also called client/server FS.
- Access through network is possible (mount).
- The file system work load is not distributed.
- Examples: NFSv2, v3, and v4, CIFS/SMB (Samba), etc...

iii **Distributed File System**

- Real file system distribution; data and metadata on different machines.
- Work load balancing (or distributed computing)
- Data could be stored in any machine in the cluster and could be accessed from any machine.
- Examples: Lustre, GlusterFS, WAFS, GFS, pNFS (NFSv4.1), Ceph, etc...

Again the focus of this project is on the third types of file system: **Distributed File System (DFS).** DFS is a good solution for huge systems (cluster of systems) which involves lot of data storages from gigabyte to petabytes of data. Normally DFS works on top of local file system and/or network file system. There is some confusion between the meanings of Network File System versus Distributed File System as both of them work over the network. The major difference between them is that Distributed File System actually distributes or splits the metadata and the actual data in to different machines (servers) which is not the case with the Network File System [11]. The other major difference is that DFS runs on more than one machine to distribute or share the workload while NFS runs on a single server. The NFS main advantage against the local file system is that it can be mounted or accessed from remote machines (clients) via the network. However, it doesn't distribute or share the file system workload among cluster of nodes like DFS does.

File system architecture advances from the first architecture which was designed to work on a single machine, then to single machine plus use of external storages - and to a single machine plus working in a network: like using NAS and SAN - and then to latest distributed file system which involves a lot of machines even heterogeneous by virtualizing the storages and by splitting the management of metadata and the actual data or file [11].

The high demand of size, performance, scalability, and reliability of file system obliged designers to seek a better solution. The latest new idea emerged uses object-based storage device (OSD) instead of block-based and even server-based

storage device unlike NFS, SAN and NAS. One of these kinds of newest Distributed file systems is called Ceph. It is open source software which was developed by Sage A. Weil in 2007. As per the founder of the software, ceph is capable of managing many petabyte-scale storage clusters with high performance, scalability and reliability [12, 24, 11, 1]. In this paper this claim is going to be investigated and evaluated.

Chapter 2 (The background chapter) discusses more about what file system is all about, its development, and about the local, network, and distributed file systems.

1.3 Ceph Overview

Ceph is an open source distributed, parallel, and network file system developed by Sage A. Weil in 2007. It is designed to be extremely scalable (gigabytes to petabytes of data) with excellent performance and reliability. Ceph is able to make maximum separation between metadata and actual data to be in different machines which helps it to be very fast in performance and highly scalable. Ceph creates an abstraction of a single file system which works on several (or distributed) servers. The other very helpful architectural approach that helps ceph to be excellent distributed file system in all scalability, reliability, availability and performance is that it uses object-based storage device **(OSD)** instead of block-based or server-based storage device which all NFS, SAN and NAS are based [12, 24, 11, 1].

Main concepts which differs ceph from other distributed file systems are:

- Maximum separation of actual data and metadata
- Independent and dynamic metadata management
- Intelligent OSDs are reliable, autonomic, and distributed.
- Dynamic sub tree partitioning:

  - ceph is capable of creating arbitrary subdirectories dynamically and intelligently based on usage patterns.

Ceph design approach is a bit different from other DFSs in its general architecture. Most DFSs architecture includes metadata servers, storage servers, and clients. However, ceph include Monitor nodes in addition to MDSs and OSDs in order to monitor ceph. To install a cluster of ceph a minimum of three nodes (servers) are required: 1) Monitor (MON), 2) Objec-based Storages node (OSD), and 3) Metadata server (MDS). And then based on usage it is possible to add more MONs, OSDs, and MDSs dynamically without stopping the service. The number of MONs is recommended to be odd. For better result hardware requirement for MONs, MDSs, and OSDs should also be fulfilled - **MONs**- normal; **MDSs** - very high RAM size, very fast CPU, and fast network; **OSDs** - very big disk size, high RAM

and fast network. The other interesting part is you can configure all of them in one configuration file which is on the Monitor (MON) node and use that same config file for the rest of OSDs and MDSs. No need to go to every node and configure [12, 24, 11, 1].

Ceph kernel client has been included in the standard Linux kernel version 2.6.34 and later. Ceph is still under heavy development and hence it is not yet ready for production server except for testing or experimental purpose at least at the time of writing this thesis.

1.4 Problem Statement

Problem statement of this project is just the title of the thesis itself which is:

"Investigating the Performance, Scalability and Reliability of a Distributed File System, Ceph."

Performance: Investigating ceph file system performance mainly testing file I/O read/write speed under different storage size and different conditions.

Scalability Investigating how well ceph performs when number of clients scale up; and the storage size (file size) is expanding.

Reliability: Investigating ceph in how extent it is fault tolerant and how well it replicates data across nodes in the cluster so that it avoids single point of failure and restore data if any node fails and/or error happens.

This problem statement has been chosen because of the current challenge facing many datacenters nowadays. There is high demand of massive storage management, high demand of a file system that could handle such huge storage system with high scalability, reliability, and performance at ease; allowing parallel access to hundreds and thousands of clients.

The most common technologies currently used by many datacenters to handle such huge storage systems are NFS (Network File System), NAS (Network Attached Storage), SAN (Storage Area Network). Even though three of them are very popular and still highly in use in many datacenters, all of them have limitation when it comes to scalability and performance. Especially because of two facts: all of them work on a single server and use block based storage system. Hence, work load balance among multiple servers and hence high performance and scalability cannot be achieved [11].

Distributed File Systems (DFSs) are currently available technologies which are the best solutions for huge data processing and accessing in file systems. Mainly because of two facts: they all distribute or share the workload of the file system

among several servers (load balancing) and most of them pursue object-based storage systems. Among currently available DFS products, ceph has been chosen for investigation and evaluation in this research work because of its some unique architectural advantages compare to other DFSs [1] (see section 1.1 and Chapter 4 for detail information about ceph architecture). The other important reason why ceph has been chosen is that there is no enough documentation (almost none) which evaluates ceph; as it is, of course, very latest product which is still under heavy development and so not yet even ready for production server.

The problem should be extended to compare ceph with other DFSs which is very important and left as future work since it is beyond the scope of this project because of time and resource limitations.

1.5 Approach

The problem statement stated above needs one to investigate the scalability, reliability and performance of ceph. Ceph is a distributed file system designed for huge networked file system to facilitate processing of massive storage systems. Hence, to investigate and evaluate its scalability, reliability, and performance in its full capacity, one of course needs a very big datacenter in petabyte scale and even more.

So; best way of investigating ceph is to benchmark its performance, scalability, and reliability up to its maximum capacity till its limitation is reached. Unfortunately, a datacenter which scales up to petabyte of storage is beyond the scope of this project. However, a cluster of ceph could be installed on a small datacenter having terabyte of disks on storage nodes so as to have as many terabytes as possible. Hence, ceph is not going to be investigated in its full capacity in this project as there are resource and time limitations. However, with available resources and time, ceph performance and scalability can be investigated up to some terabyte scale and then see how promising it is.

The other limitation is that ceph is still under heavy development. Even if one gets enough resources and time, it's hard to fully investigate and evaluate any software which is under heavy development due to facing unfixed bugs in the software. So; we can only see how much ceph is promising in this project and leave the rest for future work.

The other best way of evaluating ceph is comparing its performance and scalability with other similar DFSs. However, since distributed file systems setup, installation, configuration and benchmarking are somehow a complex task, it is again beyond the scope of this project. It is hard to setup and install more than one distributed file systems and then measure performance, scalability, and reliability in a semester project work. Hence; comparing ceph against other DFSs (like: Lustre, GlusterFS and pNFS) is left as a future work.

Due to the above reasons, the possible and realistic approach chosen to investigate ceph is as follows:

- Setup a cluster of ceph with available resources (network infrastructures, some servers terabyte of disks). See Chapter 5 (Section 5.1 for detail hardware used).
- Follow the author of ceph recommendations to setup, install and configure ceph.
- Create as many client machines as possible to mount ceph and benchmark it.
- Choose better benchmarking tool in order to benchmark its performance, scalability and reliability.
- Ceph performance, scalability, and reliability are going to be measured as follows:
  - **Ceph Performance:**

    Even though performance is a broader concept, file system performance measurement is mainly about how fast the file system is able to write and read. Accordingly and as per the description of the problem statement above, ceph performance will be measured. The plan is to measure read/write speed of ceph DFS using a benchmarking tool from different number of clients and with different storage sizes (for example: from 1 client, 10 clients, 20 clients, etc. . . and storage size 500GB, 1000GB, 1500GB, etc. . . up to 3000 GB or 3TB). Since ceph is a DFS designed to handle thousands of clients at the same time, the performance test may be required to scale up to thousands of clients and more which is beyond the scope of this project. Hence, ceph is not going to be investigated in its full capacity in this research alone due to time and resource limitation. Therefore, with the above performance test we can only see in how extent ceph is promising.
  - **Ceph Scalability:**

    The scalability of ceph is going to be investigated by answering how much its performance will be affected when the number of clients scales up; and when ceph DFS gets expanded in number of nodes and storage size in the cluster. Here, again, resource and time limit us from measuring ceph file system scalability up to its full capacity.
  - **Ceph Reliability:**

    Ceph reliability is going to be investigated by deliberately failing one, two or more of the nodes in the cluster depending on the cluster size and see if it restores the data. This, especially, helps us to prove one fact which is claimed by ceph that node failure is a norm rather than an

exception [12].

The three major limitations:

As discussed above, ceph performance, scalability and reliability is not going to be investigated in this paper in its full capacity due to three main limitations. Those limitations are:-

i **Resource and time limitation.**

This is a master thesis project which is limited by both resources and time. It is even hard to say that there is resource limitation. Because; even if enough resource is available, it is impossible to setup a very huge data center of having many petabyte scale storage nodes with a lot of metadata servers plus some monitors; and then test it with hundreds or thousands of client nodes. However, to investigate how promising ceph is in its scalability, reliability and performance, the available resources and time are more or less enough.

ii **Ceph is under heavy development.**

This is another obstacle which limits us from investigating ceph in its full ability and capacity. Ceph is under heavy development and surprisingly while working on this project within three months, ceph version has been upgraded six times and it's still under heavy development. The current version is v0.27 as of April 23, 2011. As per ceph roadmap the next version (v0.27.1) will be released after 6 days, and again the next version (v0.28) will be released after 7 days, etc. . . [1]. That's why it is not yet stable and can't be trusted with important data while it is under intense development. However, this doesn't stop us from benchmarking ceph as many datacenters out there are waiting for a better product and want to see how much promising it is so that to consider it as one choice for their file system use.

iii **The nature of the file system.**

This is almost described above (in the resource and time limitation). Due to the nature of its wide range of application which is designed for many petabyte scale; it makes it hard to investigate it fully. And, due to the distributed file systems (DFSs) complexity, it is also hard to compare it with other types DFSs within short period and limited resources.

The type and specification of hardware used in this project and the network topology are stated in Chapter 5 (Experiment setup Chapter, section 5.1).

1.6 Research Goals and Contributions

As the technology advances, the number of computer and internet users increase exponentially. Data centers are also increasing rapidly day by day. Companies and all computer users' works are highly associated with plenty of important files and data. All companies' works are highly dependent on those important files and data. If there are files, there are also file systems or file management in order to organize the files hierarchically, allocate storage for each file, maintain and keep control of access to the files, etc. Hence file system is a very fundamental part of all computer systems, especially for Linux systems its meaning and importance is enormous as almost everything is seen as a file.

We can easily imagine the importance, the high need and demand of file systems with high availability, reliability, scalability and with excellent performance. Researchers and designers of file systems have been doing great jobs to fulfill the need of all times.

Different types of file systems have different ways of doing the same job (i.e. organizing files and data in different directories hierarchically or handling directory tree management, allocation of disk spaces or partitions according to administrator's requirements, access controls, and maintaining files writing/reading/executing, etc. . .). Their performance, reliability and scalability also vary depending on their different architectural approach. Ceph also has its own approach and architectural design to tackle the high need of file system performance, scalability and reliability. As it is explained in the Motivation section above (Section 1.1), ceph has introduced some special and unique architectural concepts to improve the traditional approaches.

In many companies today's data centers there are still very high demand of file systems which can handle very big size of storage with excellent performance, scalability, and reliability [22] . Ceph is one of the latest innovated distributed file systems to meet these demands which need to be tested in order to know how well it performs the file system work. It is still under heavy development and not yet ready for production at the time of this thesis writing. Hence, there is no enough documentation about its installation, benchmarking, and evaluation. That's why this project is important.

1.7 Thesis Structure

Chapter 1: Introduction chapter

- **Section 1.1:** talks about the author's motivation to work on this project.
- **Section 1.2:** talks about overview of file systems and their types in general.
- **Section 1.3:** describes ceph in brief. It is discussed in detail in Chapter 3.

- **Section 1.4:** states the problem statement of this project.
- **Section 1.5:** is approach section how to deal with the problem statement.
- **Section 1.6:** is about the research goals and contributions.
- **Section 1.7:** shows how the thesis is structured.

Chapter 2: discusses about file system types and developments in brief including discussing about the most common file systems from each file system group (local, network, and distributed file system types). It also explains about file system performance, scalability and reliability; what they mean and how they could be measured. Finally, it talks about IOzone benchmarking tool.

Chapter 3: is all about ceph in detail; its architectural advantages, how extremely it is scalable and reliable while maintaining excellent performance.

Chapter 4: is about methodology which explains how a cluster of ceph could be setup, installed, and configured supporting with examples and sample executions.

Chapter 5: shows the result of different benchmarking of ceph.

Chapter 6: shows the result of different benchmarking of ceph.

Chapter 7: explains the analysis of the results found in chapter 5.

Chapter 8: discusses about the results and analysis observed during the experiment.

Chapter 9: gives the final conclusion of the whole work done in the research.

Chapter 10: discusses about Future work.

Appendix A: talks about why pNFS and Lustre are dropped from comparing them with ceph and discusses about pNFS installation.

Appendix B: shows sample IOzone runs and outputs.

Appendix C: shows examples and sample runs of some commands; and configuration files used in the project.

Appendix D: the three Perl scripts used in the process of benchmarking ceph:

Appendix E: supplementary graphs and boxplots of ceph benchmark results.

Appendix F: Acronyms used in the thesis.

Chapter 2

Background and Related Works

Before going direct to ceph installation, methodology and benchmarking, one need to have background about what file system is in the first place, its developments, its types, and similar works done so far. This Background and Related Works section is organized in a way that one could get enough information about file system types and their development, file system performance, reliability and scalability. Finally it discuses about how file system is tested and evaluated using a benchmarking tool.

2.1 File system types and their development

File system is system or mechanism that facilitates a lot of files and/or directories to be saved on a single partition of fixed disk and/or removable media. It is also software that supports the I/O infrastructure of operating system [13]. It, in general, handles everything related to files - organizing it in a hierarchy of directories, keeping every info about the actual data or file which is the metadata of files - owner of the file, creation date, modified date, access control for read/write/execute permission, etc. . . [17, 26]. However, the focus of this project is not to discuss about the structure and meaning of Linux file system directory tree (or about hierarchy of directories) and/SSor about metadata of files. Rather it is about the other meaning of file system which is called Type of File System or Physical File System and their performance, reliability, scalability and availability. Remember that file and file system is the very essential or fundamental part of all systems, especially for Linux, as almost everything in Linux is a file [17].

According to the second meaning of file system; we can group them into three: **Local File System, Network File System,** and **Distributed File System** based on the scale of the file system services and its distribution over the network [11].The most common types of file systems in each group are discussed in brief below.

Since the invention of UNIX Operating system in 1970s its file system called the Unix File System (UFS) has been passing through many changes to fulfill the need of its time. It is still under evolution within the BSD community [34] to meet the current and future need of high performance, scalable and reliable file system.

Many researchers and designers have put their effort in order to improve the structure or architecture, performance, scalability, and reliability of file systems to satisfy the increasing demand of storage efficiency, scalability and reliability [11]. So; achieving high performance, scalability, and reliability of file system is so much important for the digital world.

2.1.1 Local File System

Local File System is designed to work only in the local host and for the local host. It also resides in locally attached disk only [11]. Its design has been highly influenced by the very first UNIX file system (FS) for almost the last three decades [11]. Local file system is the very fundamental part of all systems. Both Network File System and Distributed File System also work in coordination and in collaboration with Local File System. Or, in other words, all local disks must be formatted with one of local file systems (ext2, ext3, reiserFS, XFS, btrfs, etc...) before used by either network or distributed file systems. Hence; the development and advancement of local file system is very essential for both local and networked file systems.

Local File Systems have gone through many changes and developments to meet the need of its time. Starting from the very first UNIX file system called FS with comparatively very low performance, with limitation in file system sizes, etc...it reaches to today's fast, highly reliable, and scalable file systems like, ext2, ext3, ext4, JFS, reiserFS, XFS, etc...And, all of them except ext2 file system support journaling. If a file system supports journaling, it keeps track of every current transactions executed against the file system as a log in a dedicated area in the file system. This journaling system helps the file system to recover without data loss in the case of sudden system failure (mostly in sudden power off). However, journaling also affects the performance of a file system. That is why ext2 is better in performance compare to both ext3 and ext4 [13].

In this section we will briefly discuss about some of the very popular, commonly used, and/or latest UNIX/Linux local file systems: FS, FFS, Minix, ext, ext2, ext3, reiserFS, XFS, JFS, and btrfs [31, 2].

i **The first Unix file system - FS**

Most of the ideas innovated at the time of this very original UNIX file system are still under use. Here below are the most common concepts of FS [34].

- FS uses partition disk to reside and does its job. Every partition disk has its own file system.

- It supports 512 byte of data blocks. So; FS divides the hard disk into 512 byte of data blocks.
- It uses the idea of superblock and i-nodes to maintain information about every file and about the file system itself.

ii **Fast File System - FFS**

FFS shows high improvement against FS which was developed under Berkeley Software Distribution (BSD). The most important developments against FS are:

- Block size challenge was tackled by using data block fragmentation under usage of 1024 and 4096 bytes of block size to avoid disk space wastage.
- Dividing hard disk into cylinder groups and metadata spread.
- Additional features:
  - File locking
  - Symbolic linking
  - Long file names: up to 256 bytes long
  - User quotas
  - Renaming files with a single system call [34]

iii **The first Linux file system - Minix**

Minix (which is to say: "mini-UNIX", UNIX-like operating system) was originally designed or written from the scratch by Andrew S. Tanenbaum in 1980s. Minix source code was firstly released to public in 1987 via his book called "Operating Systems: Design and Implementation". However, it has got its BSD License as free and open source software since April 2000. The file system used by MINIX Operating System was also called MINIX file system.

When Linus Torvalds developed the kernel of Linux operating system in 1991 he firstly used MINIX file system. Since Tanenbaum main purpose when he designed Minix was just for educational purpose, the MINIX file system had some limitations and had low performance. Some of the limitations MINIX imposed on the file system are: Very low file system size - maximum 64MB, limited length of file names - a maximum of 14 characters). And its performance was low [26].

iv **Extended File System - Ext**

Extended file system was developed by Remy Card in April 1992 and was the first file system specifically designed for Linux operating system [39]. A lot of improvements have been made by ext compared to MINIX file system

especially in the limitation imposed by MINIX file system on file system size, file name length, and in its overall performance. The file system size improved from 64MB to 2GB and the file name size improved from 14 characters to 255 characters. The integration of Virtual File System (VFS) into the Linux kernel helped the creation of Ext file system at ease and the overall performance was improved highly [17].

v **Second Extended File System - Ext2**

Ext2 is an enhancement of the extended file system developed by Remy Card. It was designed by Wayne Davidson together with Stephen Tweedie and Theodore Ts'o in 1993. Ext2 filesystem has been the choice of many Linux users since it was developed till today as it has improved many aspects of file system compared to the original extended file system (like improved disk layout and extended size limit to 4 TB plus file system size increased to 16 TB [31, 26, 13]. Its reliability, stability, and performance are excellent even in today's standard. The efficient design of ext2 which is journal-free and hence low overhead stemming has been excellent in performance. Almost always it has been the fastest file system. It also easily accommodate future updates compared to the first extended file system. [13].

vi **Third Extended File System - Ext3**

This is again an enhancement of Ext2 which was designed by Stephen Tweedie. Almost everything is the same as Ext2 except that Ext3 supports journaling [31, 13]. Hence, if any one needs to update his file system from Ext2 to Ext3, it is very easy. The very common command which is used to do the conversion is 'tune2fs'. There is no need of formatting and data migration to upgrade Ext2 file system to Ext3. Since Linux kernel version 2.4.13, it has been included as one the standard file systems in the operating system. [31]

When performance is concerned still ext2 is better than ext3 file system. Because Ext3 support of journaling imposes additional overhead to the file system [13].

vii **ReiserFS**

ReiserFS was developed by Hans Reiser. It supports metadata journaling and uses B* balanced Trees files and directories management. It is mainly popular for its high performance in small files. It also supports dynamic disk I-node allocation. Since Linux kernel version 2.4.1, it has been one of the standard file systems in Linux distributions and the first journaling file system included in the standard Linux kernel. Its main design goals were: very high performance, handle large directories, journaling. It also exceeds ext2 in performance. Reiser3 is mostly common version of reiserFS and now it is upgraded to reiser4 [31, 13].

viii **Journaling File System - JFS**

As the name implies, JFS is a journaling file system which was designed by IBM in the year 2000. It has been included in the standard Linux kernel since 2.5.6. JFS has several online utilities or tools for ease maintenance to maintain its functionality without being offline [13].

ix **XFS**

XFS was designed by Silicon Graphics, Inc (SGI) in the year 2000. It is designed to be very high-performance journaling file system and manage extremely large file systems to replace EFS (Encrypting File System). It is a 64-bit file system. XFS has been universally available since the mainline Linux version of 2.4. It is one of the oldest journaling file system for Linux and it is the standard file system for SGI [13].

x **B-tree file system - Btrfs**

Btrfs (commonly pronounced as 'Better or butter file system') is one of the very latest Linux file systems which is developed by Oracle and is licensed under the GNU General Public License (GPL). At the time of this thesis writing btrfs is under heavy development and open for contribution from anyone. fsck tool is not yet ready to be used for btrfs to fix errors and hence not recommended to use it for production [2].

Mainly it is known by its advanced copy-on-write feature which is designed to be fault tolerance which increases the file system reliability. In addition to that, btrfs design focuses on managing very large storage and ability to detect and repair errors. It has been added in the standard Linux kernel as of version 2.6.31 [2]. Hence, it is also one of the choices of file systems in the latest Linux operating system distributions.

In the installation of a cluster of ceph, btrfs is recommended for the local file systems for the OSD servers [12, 1].

2.1.2 Network File System

Network File System is a file system that can be accessed through network from remote hosts. One should not be confused with Distributed File System as in this case the metadata and the actual data are not distributed in different hosts and it work load is not shared by different hosts. It just gives file system services through the network for clients. Since it is a client-server-based application it could also be defined as client-server file system. The server exports the file system to several clients through the network. Since the clients can access and process the data stored in the server they assume the file system as if it is on their own machines.

Network or client-server file system has been a solution for the high demand of

sharing storage and data resources in networked hosts. It also facilitates the creation of the most widely used and highly scalable Network attached Storage (NAS). However, the client-server or network file system architecture uses a single server to handle the whole work and it proves its limitation in scalability and performance. That's why Distributed file system is very essential to get better result in scalability and performance [11]. See the Distributed File System section (section 2.1.3) for more information.

The most popular and widely used file systems in this category are: NFS and CIFS/SMB

i **Network File system - NFS (protocol)**

NFS is a network file system protocol which is client-server model was designed by Sun Microsystems in 1984. It is very popular and widely used network file system worldwide till today [11, 20]. The most inspired architectural advantages of NFS are [20]:

- Sharing files or data via network with high performance by using local caching and read-ahead system.
- NFS transparently exports file system from the server to clients via mounting so as the clients think as if the file system is in their local hosts.
- It supports heterogeneous machines: clients and servers should not necessarily have the same hardware and/or operating systems.
- It supports diskless workstations by booting from the network.

NFS drawbacks [20]:-

- NFS uses UDP for its transport and hence it is stateless. As a result it doesn't support all UNIX file system controls. It doesn't maintain the state about the clients.
- Migration transparency is not supported. Clients must know if the central server resource moves to another place,

ii **Server Message Block (SMB), CIFS or Samba**

SMB also known as Common Internet File System (CIFS) similar to NFS is a client-server application. It is originally designed by IBM and then highly modified by Microsoft Company in 1990's and it also suggested to be renamed **Common Interne File System (CIFS)**. Microsoft's main reason to rename it to CIFS is [18]:-

Common: to say that it is commonly available or commonly used, as CIFS can be available through the network for not only MS operating system but also non-MS operating systems like Linux.

Internet: CIFS application is not only LAN based as NetBIOS does, but it scales to large networks - Internet based by using Domain Name System (DNS) for name resolution.

File System: CIFS allows remote hosts to share files or data through a network and hence a file system of a server can be accessed by multiple clients through a network.

SMB has upgraded two times up till now by the Microsoft Company: - SMB 2 or SMB 2.0 in 2006 with Windows Vista and SMB 2.1 in 2008 with Windows 7.

The Linux version of CIFS/SMB is called **Samba** which is called Windows SMB/CIFS fileserver for UNIX (see samba manual page for detail).

2.1.3 Distributed File System - DFS

Most commonly distributed file system is interchangeably defined as network file system. It is true that it works through the network. But to define DFS more accurately: it is a super file system which abstracts a single file system by uniting and administering lot of 'distributed' file systems in which files are stored in heterogeneous or homogenous machines together but could be accessed through that abstract file system called DFS [37]. Distributed File System (DFS) main target is to achieve scalability and load balancing which has been a challenge for all Network File Systems (NFSs) for years due to single server bottleneck in case of trying to access a lot of files from many clients [40, 12].

The increase demand of huge storage systems are most commonly and widely addressed by implementation of Network File System (NFS) together with Storage Area Network (SAN) and Network Attached Storage (NAS) by many companies in their data centers worldwide [10]. However, both of them have limitations due to their dependant on a single server which as result affects the file system's scalability and performance. Distributed File Systems are best solution to distribute the work load among multiple servers and as a result a very high performance and very high scalability could be achieved with extremely parallel access at ease [11].

Some of the very popular, open source and/or latest distributed file systems are discussed below.

i **GlusterFS**

GlusterFS is one of the most popular, distributed and parallel file systems developed by Gluster Inc. It is an open source under GNU GPL license which is also POSIX-compatible [10]. GlusterFS is a client/server based system in which clients also play significant role for its functionality (like: volume management and file replication) [10, 7]. GlusterFS is mainly designed to scale

NAS storage system. It works over cluster of storages consists of a lot of storage nodes up to several petabytes level and capable of handling thousands of clients like many DFSs do [7].

GlusterFS architectural approach doesn't follow similar architectural concepts as many other distributed file systems follow. It doesn't split actual data and metadata as most DFSs do, rather it uses hashing algorithm to map file names to storage nodes. Hence, GlusterFS server is simple which just exports existing local file system (ext3, ext4, XFS for example). Storage servers run glusterfsd daemon in order to export their local file system as volume to the GlusterFS and an abstraction of a single huge file system or storage cloud of a single global namespace will be created. Data replication is handled at the client side. Involved node types are just data server and client only. GlusterFS supports both Infiniband RDMA and TCP/IP to cluster the storage servers [10, 7].

GlusterFS also gives direct access to files in different nodes in the cluster via NFS/CIFS/SMB. Its design is based on file level to store data using astackable user space translators. The normal or standard and recommended way of mounting GlusterFS from client machine is through FUSE interface. Hence client system needs to support FUSE. It is user space file system functionality in Linux OS. If FUSE is not possible, GlusterFS also supports NFS or CIFS/SMB by re-exporting the GlusterFS via NFS or SMB [7].

It is possible to scale up the size of GlusterFS volume or achieve scalability by adding additional server nodes or else by adding storage disks (bricks) from any server in the cluster without stopping the service just by running very simple command on the command line [7].

The main architectural difference between ceph and GlusterFS is that GlusterFS doesn't split metadata from actual data unlike ceph uses separate metadata servers to handle files in the cluster storages and to facilitate access to clients.

ii **Google File System - GFS**

GFS has been designed by Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung to meet the need of fast growing demands of Google's data processing. It is a distributed file system for data-intensive applications. It is fault tolerant when running on inexpensive disk or hardware and delivers high performance for a lot of clients [33]. Even though it shares many ideas and goals from the previous distributed file systems, it also adds some improvements to tackle the current and anticipated challenges:-

- Failures of any component are the norm (not exception) Failure could happen by different causes: - it could be human errors, operating system or application bugs, disk failures, networking or power failure, and so on. Hence; to resist those failures and errors GFS is designed to integrate

the following key concepts in to the system: fault tolerance, continuous monitoring, error detection, and then automatic recovery.

- Reconsidering the traditional design assumptions for I/O operation and block sizes from Mutli-GB sized file norm to KB-sized files inclusive.

GFS is mainly designed to meet the demand of Google itself. The main architectural approach is that GFS uses cluster of machines with a single master and multiple chunk servers so as accessed by multiple clients. The master maintains the metadata of the file system. For the first time client contacts the master and gets information from the metadata about chunks stored in chunk servers and then for next time the client will directly contact the chunk servers [33, 37].

The GFS chunk concept is similar to blocks in traditional file system except that its size is much larger than typical file system block size which is 64MB. Hence, by having big chunk size it increases read/write performance of the file system. The idea is that as the chunk size increases number of chunks decreases and hence small metadata for the chunks in the master server which increases performance. However, it has its own drawbacks in case of many small size files [37].

GFS design approach is different from ceph even though both of them are designed for huge storage systems and to allow thousands of clients to access the file systems in parallel. For example, GFS has a single master metadata server while ceph has many metadata servers (MDSs) as one wants depending on the size of data and/or the size of the file system. GFS uses chunks as its efficient storage approach while ceph uses Object-based Storage Devices (OSDs) to store data. GFS could be best for Google Company itself, but for other organizations having a lot of small sized files, GFS may not be a good choice.

iii **Lustre**

Lustre is a distributed file system which was originally developed in a research project in Carnegie Mellon University (CMU) in the year 1999 [36, 10]. However it is now open source under a GNU General Public License (GPL) and POSIX compliant which is owned by Sun Microsystems Inc. Lustre is one of the most popular extremely parallel distributed file systems which has been used by many HPC High Performance Computing Centers in the whole world, especially in the oil and gas industries, in manufacturing, finance sectors, and the like [27, 16]. The name Lustre is derived from **L**inux Cl**uster**.

Lustre and ceph have similarities and differences in their architectural approach. Both of them split metadata from the actual data. Both of them also use object-based storage devices (OSDs) for their cluster systems storage solution. Lustre architecture is composed of Metadata servers (2 MDSs - one metadata target and one failover), Object Storage Servers (OSSs), and clients.

Both Lustre and Ceph are also extremely scalable as per [1, 16, 40].

Lustre's main advantages are also similar to ceph [1, 10, 16].

- It is highly scalable.
- It has high parallel performance.
- It can handle thousands of files because of having good file I/O performance.
- Separation of metadata and actual data
- It has reliable OSDs.

Lustre's main differences with ceph distributed file system are[12, 10, 16]

- Lustre uses single metadata server plus one failover to handle file system designed for hundreds and thoursands of clients with a lot of OSSs and Storage Targets (OSTs). So; there is limitation in the number of metadata server which limits its scalabilty as well. Ceph doesn't impose limitation on the number Metadata Servers (MDSs). It supports multiple MDSs depending on file system size.
- Lustre assumes its Object Storage Servers (OSSs) are reliable. Hence, for data redundancy, it is recommended to use hardware RAID system to protect against disk failure. However, the Object-based Storage Devices (OSDs) of ceph are reliable and intelligent which assumes node failure as a norm. It stripes data across storage nodes to replicate data and recover itself in case of disk failure.

iv **pNFS - NFSv4.1**

pNFS (acronym for parallel Network File System) is very young distributed file system which is designed as an enhancement to the NFS (network file system) protocol to eliminate the performance and scalability challenges facing NFS. The new pNFS - NFS version 4 minor version 1 (NFSv4.1) parallel distributed file system has got official approval from Internet Engineering Steering Group (IESG) and received its Request for Comments (RFC) number in January 2010. It is still under heavy development and not recommended for production use [6, 10, 15].

pNFS, similar to ceph and Lustre, is able to split the metadata from the actual data. NFS file system has limitation of scalability and performance due to the fact that every process of the file system work is done by a single server. However, using pNFS it is possible to distribute the work load among multiple servers and it is also possible to access files in parallel for many clients at the same time. Similar to ceph and Lustre, pNFS cluster design also consists of Object-based Storage Devices (OSDs), Metadata servers (NFSv4.1 Servers)

and pNFS Clients [6, 35].

pNFS, of course, achieves huge performance and scalability advantages over NFS, especially due to allowing parallel connection between clients and data servers by keeping the metadata server out of the data path [35]. The interesting advantage of pNFS is that in addition to benefiting from the parallel access for many clients to storage nodes it also keeps the standard NFS protocol by just upgrading the NFSv4 with minor version 1 [6].

Comparing pNFS with ceph: [6, 10, 35, 15]:

- Both achieve separation of metadata and actual data
- Both allow parallel access for clients to data servers directly.
- Both of them use object based data storages which is reliable.
- pNFS currently allows only one metadata server which affects its scalability. Ceph allows multiple metadata servers.
- No dynamic subtree management in pNFS which is the case with ceph.

v **Ceph**

Ceph, as it is the focus of this research, is one of the very latest object-based parallel distributed file systems with POSIX semantics designed to be extremely scalable and reliable with excellent performance [12, 1]. As it is stated in the Introduction chapter (Chapter 1: section 1.4), ceph was developed by Sage Weil as part of his PhD research at University of California, Santa Cruz. It is still under heavy development and hence not suitable for production server except for benchmarking and experimental purpose at least at the time of this research.

The name ceph doesn't follow the acronym trend that many other file systems have followed. See the other file systems' acronym use above. The name ceph is related with Cephalopods which is metaphor for the concept of distributed file system [4].

The standard distribution of Linux kernel version starting from **2.6.34** includes ceph kernel client. The current ceph's latest version released is **v0.27** as of 23 April 2011 [1].

Compared to other Distributed File systems, ceph has some unique architectural advantages.Ceph and its architecture is discussed in detail in the next Chapter (Chapter 3) and its installation and configuration is discussed in Chapter 4.

2.2 Benchmarking a file system

File system performance, scalability, reliability and availability could be tested, measured and evaluated using different benchmarking tools. Since there are quite a number of different open source file system choices under UNIX/Linux families and since cost is not an issue, the only main decision criteria to choose from those lists of file systems are mainly depend on where they stand in regard to performance, scalability, reliability, and availability.

Some of the benchmarking tools widely used are bonnie++, IOzone, LMbench, PostMark, Hbench, IOMeter, etc ... [31, 32].

IOzone is chosen to measure and evaluate the performance, scalability, and reliability of the distributed file system, ceph. IOzone is discussed in the Experiment Setup Chapter (Section 2.2.4) in detail including the reason why it has been chosen.

2.2.1 File system Performance

File system performance is best measured by the amount of time it takes to give service to clients [36]. And, the main services a file system provides to clients are data or file write/read, and maintaining the data. So; how fast a file system can read and write is the vital metrics for its performance. If it takes shorter time, then the performance is higher; commonly measured by KB/sec. Hence, bigger is better.

File system performance is affected by scalability, reliability, and availability. A file system performance will be reduced at least to some extent in order to incorporate scalability, reliability, and availability all together [74]. That is why, for example, ext2 file system is always better in performance compare to the advanced ext3 and ext4 file systems [13]. It's all because ext3 and ext4 incorporate journaling.

We can divide file system performance in to two major categories: **Meta-data** and **User-Data**. The **Meta**-data performance has nothing to do with the speed of writing on or reading from the actual data rather that could be categorized to the **User Data** performance. The **Meta-data** performance could be evaluated by measuring every function about the data except the actual data itself; like creating or deleting files and directories (directory tree management), opening and closing, updating file attributes (creation date, owner, access permissions, etc), disk space allocation, etc ... [32]

When we test or evaluate the performance of any file system using one of the benchmarking tools listed above we normally do not get the performance of the Meta-data and User-data separately. Almost all benchmarking tools provide a general or overall performance of a file system's I/O operations (i.e. the cumulative performance of both Meta-data User-data functions). That is, of course, one of

the drawbacks of file system benchmarking tools as it makes it hard to identify the specific part of the weakness and/or strength of I/O operations of file systems [32]

2.2.2 File system scalability

Scalability can be seen in wider range. Scalabilty could be in terms of performance, it could be in terms of availability, it could also be in terms of number of access it provides in parallel [10]. It depends on the interest of the investigator.

In this particular project, file system scalability is measured in terms of performance (i.e. to how extent ceph DFS performance is affected when the number of clients using the file system increases both in number and in file size they use; or when the file system is expanding in size). It is almost natural to expect performance reduction at least to some extent when file size, storage, and number of clients are getting increased (scaling up). However, for a file system claimed to be extremely scalable (as ceph claims), its performance should not be affected significantly or it should be in acceptable level even though its size and number of clients fluctuate highly [36].

2.2.3 File system Reliability

Data safety is one of the very essential requirements of file system users. Hence, file system reliability is very crucial value. File system reliability is measured by investigating its error detection and recovery mechanism (or its fault tolerance), its data replication or redundancy system.

In this project ceph reliability will be measured by deliberately failing one or two OSD nodes and then investigate its recovery system.

2.2.4 Benchmarking tool: IOzone

IOzone is an open source benchmarking tool originally proposed by William D. Norcott (from Oracle) and developed by Don Capps and Tom McNeal (from Hewlett Packard). IOzone is designed to work only for file system performance tests. It doesn't work on raw disk that doesn't have a file system [5].

IOzone's main popularity comes from its nice and attractive creation of 3D graphics which works in conjunction with gnuplot. Gnuplot is a portable command-line graphing utility for Linux that can generate 3D graphs or plots of a function in addition to just two-dimensional [5, 3].In addition to that iozone has a lot of interesting options which is discussed below.

i **Why IOzone?**

There are, of course, other similar benchmarking tools which can do similar work. Benchmarking a file system and some of similar benchmarking tools are mentioned in section 2.2. In general, IOzone has the following especial features and advantageous [29, 19, 5] compare to the other available benchmarking tools. And, that is why it is chosen to benchmark ceph distributed file system.

- It works for all types of file systems (local, network, and distributed file systems).
- It is easy to use and it works under many platforms (or operating systems) which includes Linux and Windows.
- It assumes its execution is bottlenecked by storage devices to avoid the significant effect of CPU speed and RAM size specifications.
- It's Compatible for very large file sizes.
- It's Compatible for multi-process measurement.
- It's Compatible for both single and multiple stream measurement.
- It's Compatible for POSIX Asynchronous I/O
- It's Compatible for POSIX Threads, or Pthreads.
- Its I/O Latency plots feature.
- Its processor cache size configurable feature.
- Excel importable output for graph generation feature.
- Compared to bonnie++, IOzone has more features and generates more detailed outputs than the common read and write speeds. It measures many file system's operations (files I/O performance), like: read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read/write, pread/pwrite, aio_read/write, and mmap 56,66.

ii **IOzone how-to:**

One can download the latest IOzone source code from IOzone website http://www.iozone.org using 'wget' command one can download one of the following latest IOzone codes of 18-Mar-2011:-

```
http://www.iozone.org/src/current/iozone3\_373.tar
```

After downloading IOzone source code, go to its folder and type **'make'** in the command line to compile it and then **'make target'** to install. Then, enjoy using IOzone.

It is also possible to install IOzone on the Linux command line by typing:

```
$ apt-get install iozone3.
```

Since the file system benchmarking result is highly influenced by the size of the system's buffer cache, before running IOzone one need to know the following rules [5]:

Rule 1: For accuracy the max size of the file going to be tested should be bigger than buffer cache. If the buffer cache is dynamic or confusing to know its size, make the max file size bigger than the total physical memory which is in the platform [5]

Rule 2: Unless the max file size is set very smaller than the buffer cache, you must see at least the following three plateaus:

- File size fits in processor cache.
- File size fits in buffer cache.
- File size is bigger than buffer cache.

Rule 3: Use -g option to set the maximum file size value. Refer manual page of IOzone command (man iozone) for more information.

IOzone Command Line Options:

For simple start use the automatic mode:

```
$ iozone -a

-a      Run in automatic mode; it generates output that covers all
tested file operations for
record sizes of 4k to 16M for file sizes of 64KB to 512MB.

See below for other important options \cite{56}:

-b filename
Iozone will create a binary file format file in Excel compatible output
of results.

-c
Include close() in the timing calculations.

-C
```

```
Show bytes transferred by each child in throughput testing.

-d #
Microsecond delay out of barrier.

-e
Include flush in the timing calculations.

-f filename
Used to specify the filename for the temporary file under test.
-g #
Set maximum file size (in Kbytes) for auto mode.

-h
Displays help.
-i #
Used to specify which tests to run. (0=write/rewrite,
1=read/reread, 2=random-read/write
3=Read-backwards, 4=Re-write-record, 5=stride-read, 6=fwrite/re-fwrite,
7=fread/Re-fread,
8=random mix, 9=pwrite/Re-pwrite, 10=pread/Re-pread, 11=pwritev/Re-pwritev,
12=preadv/Re-preadv).
One will always need to specify 0 so that any of the following tests will have a file
to measure.
-i # -i # -i # is also supported so that one may select more than one test.

-s      Sets file size in KB for the test. It also accepts MB and GB which needs to be
explicitly specified (-s #m for MB, -s #g for GB).
-R      Generate Excel report.
```

For more information read IOzone manual page.

The other very interesting feature of IOzone is for every run it gives a summary of the conditions of that particular run in its output including the command line used. See below one sample run condition in the output of IOzone test.

Sample IOzone output (only the run condition)

```
1   Run began: Sat Apr 30 17:18:32 2011
2
3   Record Size 256 KB
4   File size set to 83886080 KB
5   Command line used: iozone -i 0 -i 1 -r 256 -s 80G
6   Output is in Kbytes/sec
7   Time Resolution = 0.000001 seconds.
8   Processor cache size set to 1024 Kbytes.
9   Processor cache line size set to 32 bytes.
10  File stride size set to 17 * record size.
```

See Appendix B for more IOzone sample runs and outputs.

Chapter 3

Ceph and its architecture

As described in the above two Chapters, Ceph is one of the very recent and very promising parallel, object-based, distributed file system; especially from the point of its architectural benefits. This chapter discusses about its general architecture and its architectural advantages which help ceph to be extremely scalable and reliable, with excellent performance as per [12, 24, 11, 1]. Its installation methods and ceph 'how-to' are discussed in the next chapter (Chapter 4: Methodology). One of the main goals of ceph is to be POSIX-compatible in addition to being free software (an open source which is distributed under the GNU Lesser General Public License) [1].

3.1 Ceph General Architecture

Ceph general architecture consists of Metadata server (MDS), Object-based Storage Devices (OSDs), Monitors (MONs), and, of course, clients. A minimum of one from each type is required to make ceph start, and then ceph architecture allows you to scale up the number of MONs, MDSs, and OSDs up to thousands more according to the need of your file size [1]. The hardware requirement for each type of server in the ceph cluster is found in the ceph methodology chapter (chapter 4).

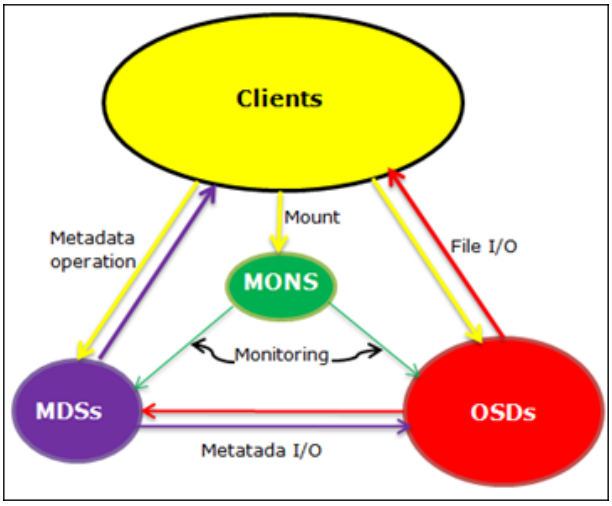


Figure 3.1: Ceph General Architecture

Figure 3.1 shows the general architecture of ceph. The ceph cluster representative is the MON node(s). One typical example of client and ceph file system interaction is described below after the client node mounted the ceph file system via MON IP-address:-

- Client --------> MDS (open request)
- MDS --------> Client (provides file inode, file size, and stripe info,...)
- Client <-------> OSDs (direct read/write from Client to OSDs)
- Client --------> MDSs (close request)
- MDSs --------> (saves the changes)

In the above ceph file system interaction with client, MON part is not mentioned. Actually, the main job of MON nodes is monitoring the whole process. The next section (section 3.2) MONs job in the ceph cluster is discussed very briefly.

3.2 Ceph Architectural Advantages

Ceph has incorporated the following very interesting and key concepts in its design and architecture in order to achieve its three main goals (i.e. extreme scalability, strong reliability, and excellent performance).

- Object-based storage instead of block-based
- Ceph PG, CRUSH and RADOS
- Maximum separation of data and Metadata
- Metadata Dynamic subtree partitioning management

3.2.1 Object-based storage instead of block-based

The difference between block-based and object-based storage system is not on the physical media they use (or type of disk drive), rather it is about their difference in the type of interface they use and the way they communicate with the disk for data I/O [36]. Block-based storage system uses the traditional way of storing data direct to fixed sized sectors (or blocks of size 512, 2048, or 4096 [21]) in the disk via logical block interface. But, object-based storage system (OSD) creates a lot of objects for the data and saves it on the disk via object interface by dividing the file system in to storage part and operating system (OS) part [9, 14]. Figure 3.1 illustrates their difference graphically.

Each object in object-based storage system consists of data, OID (Object IDentifier), attributes, and metadata. This helps OSD, unlike block-based storage, to take the responsibility of the management of data layout and maintaining the attributes of data objects on the disk from the host operating system.

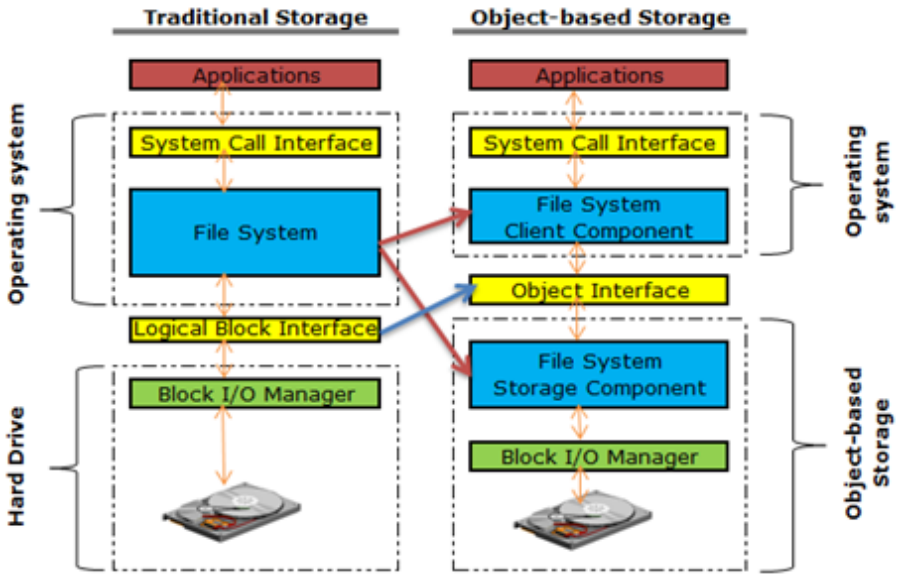


Figure 3.2: Traditional storage Vs Object-based storage.

This key concept of object-based storage system needs advanced file system compatible with communication with disk via object interface unlike via the traditional block interface way. In block-based file system the whole metadata of data in the disk is managed by the host OS. However, in object-based file system OSD takes part of the metadata responsibility out of the OS by dividing the file system into storage part (OSD) and OS part. This helps metadata server to get out of the data path so as client can directly access data via OSD. This brings huge advantage to improve a file system scalability, reliability, and overall performance [9].

Object-based storage system is relatively a new concept. Most widely used file systems nowadays, including NFS, NAS and SAN, depend on block-based storage system [11, 21, 36]. In general, all OSD-based file systems are capable of splitting metadata from actual data and hence use separate servers for the metadata and OSD servers. Metadata server mainly maintains mapping files to objects and current physical address or location of each object. And, the OSD server manages the actual data I/O operations in the form of objects and maintains each object attribute and the data layout on the disk [36]. In this way object-based distributed file systems (DFSs) allow clients to access data directly via OSD server. The metadata server is only needed in the beginning of the process when a file is opened then it will get out of the data path. This improves the object-based DFSs' scalability and performance in a great extent. And, in the contrary, block-based file systems suffer scalability and performance due to metadata bottleneck [36, 9].

Ceph has implemented today's intelligent OSD based storage system in its architecture.

- **Ceph OSD servers**

Object-based Storage Device (OSD) protocol has already become the standard storage system to pursue for the next generation storage system by replacing obsolete block-based storage systems [43, 30, 25]. And, the latest distributed file systems' use intelligent OSDs which stripe data across cluster of OSD servers so as to be reliable and fault tolerant [41]. Ceph OSDs are intelligent enough to detect errors and failure recovery. Since ceph is multi-petabyte scale parallel, network, and distributed file system, its OSDs should also be dynamic and autonomic enough to do data migration in the time of ceph cluster expansion [11, 1].

3.2.2 Ceph PG, CRUSH and RADOS

Sage Weil (the author of ceph) has developed a special algorithm called CRUSH - Controlled Replication Under Scalable Hashing [11]. CRUSH algorithm is used to replicate data objects across some specific group (PG Placement Group) from the available OSD devices. The data distribution policy follows the placement rule which is according to each OSD device weight (device's capability) and placement position. Figure 3.2.1 illustrates how data objects are grouped in to PGs and then distributed across OSDs via CRUSH [11].

CRUSH is also designed to maintain balanced usage of storage and device bandwidth by keeping the uniformity of distributing data objects across OSDs to maintain load balancing [11]. The placement rules are also defined by CRUSH; and hence strategy of the replication or distribution of data objects across PGs can be defined by system administrator. It has a command named 'crushtool' [1] which can be used to manipulate the CRUSH mapping. This command is mainly important when very big cluster of OSDs are used. By default CRUSH placed all OSDs

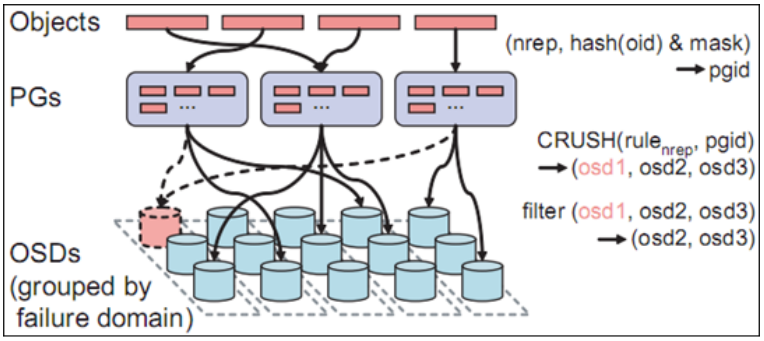


Figure 3.3: Grouping objects, PGs, and OSDs according to CRUSH rules (taken from [11]).

in a single pool which is ok for small cluster [1]. CRUSH also considers device failures as a norm via working in coordination with RADOS.

RADOS (Reliable, Autonomic Distributed Object Store) is the other key concept that ceph incorporates in its architecture. As the name implies, it facilitates reliable and high performance, load balancing through balanced data distribution across cluster heterogeneous OSD nodes while the cluster expands up to multi-petabyte scale. Its functionality works by the coordination of monitor nodes and OSD nodes together. In general, RADOS is responsible for data replication, balanced data distribution, failure detection and recovery, and creation of a single logical object store (which is a total all OSDs) with high scalability, reliability and performance [11, 23].

3.2.3 Maximum separation of data and Metadata

The biggest obstacle for the poor performance of many networked or distributed file systems which use the traditional system is the use of the same bottlenecked path for both data and metadata. Even if high storage performance is achieved; due to denial of parallel access to clients, the whole file system performance suffers [8].

As discussed above (section 3.1), object-based storage device facilitates separation of metadata and actual data by taking some of the responsibility of a file system management. Since OSDs are intelligent enough to communicate direct with clients by getting small initial help from the metadata server, it has solved the long lived obstacle of data and metadata bottleneck issue [11, 36, 9, 8].

Metadata scalability is more complex than scaling OSDs or increasing performance of storage systems [12, 8]. That is why some of the distributed file systems are unable to scale up the metadata server performance due to its complexity. For

example, pNFS(NFSv4.1) improves from NFS in scalability and performance considerable by allowing parallel access to many clients concurrently due to the face that the metadata and actual data paths are now separated and gives a great relief for a single server bottleneck issue the NFS has been facing. However, pNFS is currently using single metadata server in its architecture [35, 40].

One of the key concepts ceph implemented in its architecture is its maximum separation of data and metadata management [12, 11]. The amount of metadata workload is almost half of the total workload of a file system, and hence needs a special attention and development to get better file system performance. Ceph architecture maximizes the separation of metadata management and file data management. As discussed in section 3.2 above, ceph CRUSH feature reduces the metadata workload by letting clients calculate the location of objects of a file; rather than requesting metadata server for its location (look up). Ceph also doesn't impose on its file system performance by allowing only very few limited number of metadata servers or OSDs. Depending on the size of the ceph cluster, multi MDSs are possible with ceph architecture [12, 11]. Figure 3.2.2 illustrates how metadata and actual data are separated.

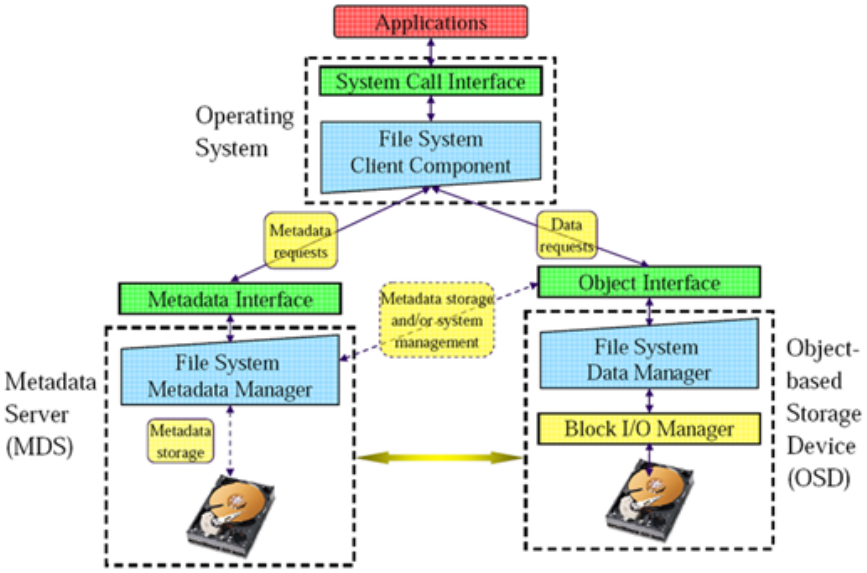


Figure 3.4: Decoupled data and metadata file system architecture [36].

3.2.4 Metadata Dynamic subtree partitioning management

This is the very astonishing key concepts developed by ceph which is almost unique and great feature of ceph. Cluster of ceph MDSs dynamically balance their load according to currently busy directories (highly accessed). At any time if some parts of file directories are extensively accessible dynamically MDSs will re-arrange their

workload and those busy directories get attended by more MDSs and then again change their positions according to the next busy directories. This feature improves performance by balancing workload across cluster of MDS servers. See Figure 3.2.3 for graphical illustration of this feature.

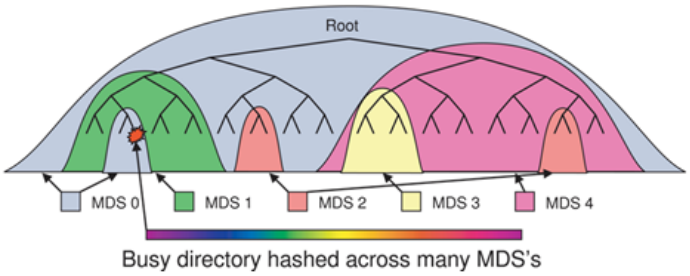


Figure 3.5: Dynamic sub tree partitioning management [12].

Chapter 4

Ceph Methodology

This chapter deals with ceph how-to. Ceph introduction, background and its architecture are discussed in the above three chapters (section 1.4, 2.6, and chapter 3). In the Introduction chapter section 4 (Approach) a summary of how to tackle the problem statement is discussed. In this chapter we will mainly discuss ceph installation and configuration in detail. Most of the installation commands in this chapter are based on Debian and Ubuntu Linux distributions.

4.1 Preparation before Installation

It is always good to prepare properly and implement the recommendation of the author of a product before going direct to installation and configuration. To install and configure ceph (a cluster of ceph) we need to know at least its minimum requirements in both hardware and software perspectives.

4.1.1 Hardware Requirement

Ceph is a distributed network file system which splits the metadata and actual data to be processed in different machines. In addition to that ceph is also designed to have other machine(s) which does the monitoring work. One can clearly understand that a minimum of three machines are required to install ceph. One for metadata server (MDS), one is for data server which is called Object based Storage Device (OSD), and one is for monitoring which is called Monitor (MON). It is also possible to install ceph on a single machine by creating a minimum of three virtual machines (VMs). However, unless it is for test, installing ceph on a single

machine using VMs has no benefit. Ceph is designed for huge data processing systems (cluster of machines) to be highly scalable with excellent performance.

The group of machines which are used by ceph to make a single abstraction of file system which is called Distributed file system is called Cluster of Ceph or simply **Ceph Cluster**. Even though the minimum requirement to setup ceph cluster is three machines, as per the size of file system or its data size, number of each part of ceph cluster can easily be scale up to petabyte size and more, according to [11, 1] which is going to be investigated in the next two chapters. Hence, the number of MDS, OSD, and MON depends on the size of one's file system and storage.

The Metadata server (MDS), Object based storage Device (OSD), and Monitor (MON) have different tasks in the ceph cluster. According to their tasks, the requirement of their hardware also varies.

i **Metadata Server (MDS)**

MDS doesn't store the actual data rather stores data about the actual data. So it doesn't need to have big size disk. However; since ceph allows parallel connection for many clients to access different files or even same file at a time, it needs to process a lot of cmds daemon instances and as a result needs a lot of RAM. In addition to that in order to fulfill its promise of many parallel connections at a time it needs to work as an intermediate between client and storage nodes. So, the MDSs must have fast network (high bandwidth and low latency). The number of MDSs depends on the size of file system and storage. Only one is possible; however 2 or more is recommended for load balancing (to share the work load) and redundancy (to increase reliability).

ii **Object based Storage Device (OSD)**

The main purpose of OSD server is storing the actual data and needs to have high disk size. It also needs to have fast network to allow fast connections to the MDS(s) and Clients as well. The RAM size requirement is a bit less than the same need by MDS server. To have better file system caching (FS-Cache) in the OSDs, lots of RAM is an advantage. Number of OSDs server again depends on the size of file system and storage. Commonly the number of OSD servers is much more than the number of MDSs as the main purpose of distributed file system like ceph is to manage a lot of data processes for many clients in big sized data storages.

iii **Monitor (MON)**

The whole cluster of ceph is managed and controlled via MON server(s). MON has one central configuration file (/etc/ceph/ceph.conf) to configure both MDSs and OSDs. This is one advantage of ceph architecture. Especially, when we have a lot of OSDs and/or MDSs, configuring all of them separately on each node would have been a tiresome work. However, since MON servers are not

Summary of Ceph Hardware Requirements						
	CPU	**RAM**	**DISK**	**Network**	**Qty**	**Remark**
MDS	Too fast	Very big	Normal	Too fast	Any	MDS doesn't store actual data. 1 is enough; 2 or more for redundancy and load balancing.
OSD	fast	Big	Too large	Fast	Many	OSD stores the actual data; so needs lots of storage.
MON	Normal	Normal	Normal	Fixed IP	Odd	MON mainly needs fixed IP as it holds the central config.

Table 4.1: Summary of Ceph hardware requirement.

involved in the direct data processing of ceph cluster, the amount of RAM and CPU requirements are less. We only need fixed network address and little disk size, few RAM and CPU speed. The number of MONs is recommended to be odd (1,3,5,...). One is, of course, enough in most cases and two is not recommended (otherwise both machines should always be). The next option will be to have 3 numbers of MONs which is ideal.

4.1.2 Software requirement

In preparation to install and setup a cluster of ceph one needs to carefully implement the recommended software needed on each machine which is used in the ceph cluster (i.e. on MDSs, OSDs, and MONs) and on client machines as well. Even though ceph is still under heavy development, its requirements for installation is not as hard as pNFS and Lustre distributed file systems installation at least as of their current status. Linux kernel configuration, compilation, and installation steps are not required for ceph unlike required by both pNFS, and Lustre currently (at the time of this research) provided that the Linux kernel version is 2.6.34 or later.

i **Ceph Client Requirement:**

Linux kernel starting from version 2.6.34 and later already supports ceph. Hence, one doesn't need to build or install ceph client software in order to be able to mount ceph distributed file system. One can directly use the following mount command to mount ceph:

```
$ mount -t ceph Monitor_IP:/ /mnt/point
```

If the kernel version used is older than 2.6.34, kernel patching is required via linux-ceph-client.git tree using git command as follows:

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/sage/ceph-client.git
```

The above git tree contains full kernel for latest ceph client code. If one needs the ceph module only, there is also a standalone tree which updates itself automatically from the main repository (linux-ceph-client.git). Building ceph client from the standalone tree is recommended by [1] because it is lightweight and hence save time and resource.

```
$ git clone git://ceph.newdream.net/git/ceph-client-standalone.git
```

And you also need to run:

```
$ git branch master-backport origin/master-backport
```

Now, following the following commands to build ceph client:

```
$ git checkout master-backport
$ make -C libceph
$ cp libceph/Module.symvers ceph/
$ make -C ceph
$ make KERNELDIR=/path/to/linux  -C libceph  # or against some other kernel
$ cp libceph/Module.symvers ceph/
$ make KERNELDIR=/path/to/linux -C ceph
```

There are two ways of loading ceph module: one is using 'insmod' command, and the other is by using 'modprobe' command. Follow the commands below:

Loading ceph module using 'insmod' command:

```
$ insmod ./libceph/libceph.ko
$ insmod ./ceph/ceph.ko
```

Loading ceph module using 'modprobe' command:

```
$ make modules_install
$ depmod
$ modprobe libceph
$ modprobe ceph
```

Then reboot your machine and enjoy with ceph client capability. Mouting ceph file system is same as above (you only need to the IP address of the Monitor (MON) in the ceph cluster and make a mount point or directory). That's all. It is also possible to use the host name of the Monitor machine instead of IP-address if mount.ceph is installed in /sbin directory.

ii **Local File System requirement:**

Ceph doesn't impose any specific local file system to be used in MONs MDSs, except recommending for the OSD servers to use **btrfs** as their local file sys-

tems [1]. **Btrfs** is copy on write file system which makes the file system more fault tolerance and its other features also make it easy to repair and administer [2]. Without the help of RAID, btrfs creates internal data redundancy which increases reliability.Journaling performance is also better with btrfs file system [1] . Using **btrfs** in OSDs facilitates easy OSD cluster deployment in the ceph cluster which provides better scalability [1]. btrfs file system is also discussed in chapter 2 (Section 2.1.1).

- **Installing btrfs file system:**

  Btrfs is discussed under Local file system section in the Background chapter (Section 2.1.1). In this section, we will just highlight some of the commands mainly used to install btrfs and commands used to convert from ext3 or ext4 file systems to btrfs.

  Install the necessary tools and dependencies for btrfs: Installing btrfs file system is quite easy. First install btrfs-tools and build-dep, and use 'mkfs.btrfs' command to create btrfs file system.

  ```
  $ apt-get install btrfs-tools build-dep
  $ mkfs.btrfs   /dev/device-name          (to create btrfs file system on the given device)
  ```

  Btrfs file system can also be created on many devices at a time using mkfs.btrfs command.

  ```
  $ mkfs.btrfs /dev/sda /dev/sdb /dev/sdc
  ```

  At any time later, more devices can also be added to the already created ones. The btrfs reliability (or fault tolerance) works in a way that btrfs will mirror metadata across two devices and will strip data across all provided devices. If there is only one device used as a btrfs file system, the metadata will be duplicated on that single device. For more info, one can read the 'mkfs.btrfs' manual page (man mkfs.btrfs).

  Mounting the btrfs file system is also straight forward once the btrfs file system is created in the device(s).

  ```
  $ mount /dev/device-name   /mnt/point
  ```

  It is also possible to convert from ext3 or ext4 file system to btrfs by using **'btrfs-convert'** command without formatting the disk.**Btrfs-convert** command uses **libe2fs** to read the ext3 and ext4 file systems.

  Here is the step how to convert from ext3/ext4 to btrfs file system:

  - Check and repair the ext3 file system first)

```
$ fsck.ext3 f /dev/device-name
```

– Convert from ext3/ext4 to btrfs

```
$ btrfs-convert /dev/device-name
```

– Mount the btrfs FS

```
$ mount -t btrfs /dev/device-name /mnt/point
```

– Mount the ext3 or ext4 snapshot and creates an image file

```
$ mount -t btrfs -o subvol=ext2_saved /dev/device-name /ext2_saved
```

– Loopback mount the image file

```
$ mount -t ext3 -o loop,ro /ext2_saved/image /ext3
```

That is all. Now, the /dev/device-name is a btrfs file system. It is also possible to return back to the original ext3 or ext4 file system after converted to btrfs. To do so; first unmount the btrfs file system completely and then using the same btrfs-convert command with '-r' flag reverse back to ext3 or ext4 file system. Here are the steps:

```
$ umount /ext3
$ umount /ext2_saved
$ umount /mnt/point
$ btrfs-convert -r /dev/device-name
$ mount -t ext3 /dev/device-name /ext3
```

(See sample use of mkfs.btrfs command in Appendix C.)

iii **Allowing password-less root ssh login:**

Ceph central configuration file is located in the Monitor (MON) machine and after configuration, the mkcephfs command is run on the master (MON) node to make ceph file system (see also **Section 4.4**). The master node (MON) needs to have password-less ssh login access to all other slave machines (i.e. MDSs and OSDs).

Most commonly used way of allowing password-less ssh login access is setting up ssh authorized_keys by generating ssh-key and then copying the key to the nodes to which password-less access is needed. (See password-less ssh sample execution in Appendix C).

4.2 Ceph Installation:

As it is common in Linux software installation, there are mainly two ways of installing ceph in Linux machines: using source code or using the precompiled packages. If one needs simplicity the second option is recommended. If one needs very recently updated software, the first option is to go for. We will see both options in this section.

i **Installing ceph using precompiled package**

Before using 'apt-get install' command in order to install ceph, the /etc/apt/source.list needs to include URLs to the main ceph repository for the stable version.

```
deb http://ceph.newdream.net/debian/ DISTRIBUTION main
deb-src http://ceph.newdream.net/debian/ DISTRIBUTION main
```

DISTRIBUTION can be replaced with the any Debian or Ubuntu Linux releases: (like: squeeze, maverick, lenny, etc...).

Install bzip2 to make sure that the 'apt-get' command reads .bz2 compressed file.

```
$ apt-get install bzip2
```

The last, but not least, step required to install ceph is to get the public key of the author of ceph from http://newdream.net, as the packages are signed by the his PGP (Pretty Good Privacy).

```
$ wget http://newdream.net/~sage/pubkey.asc -q -O - | apt-key add -
```

The last, but not least, step required to install ceph is to get the public key of the author of ceph from

```
http://newdream.net/~sage/pubkey.asc
\end{verbatim}

as the packages are signed by the his PGP (Pretty Good Privacy).

\begin{Verbatim}
$ wget http://newdream.net/~sage/pubkey.asc -q -O - | apt-key add -
```

Now; we are almost done. First, update the system with the latest list in /etc/apt/source.list and then install ceph as follows:

```
$ apt-get update
```

```
$ apt-get install ceph
```

As ceph is still under heavy development its updates and bugfix packages are released regularly. In order to automatically update with the latest release and bug fixes, you need to add ceph git branch URL in the /etc/apt/source.list as below:

```
deb http://ceph.new
dream.net/gitbuilder-deb-amd64/debian/BRANCH/
DISTRIBUTION main
```

Where:

BRANCH can be replaced with: master, next or stable. And,

DISTRIBUTION should be replaced with the recent Debian or Ubuntu Linux releases: (like: squeeze, maverick, lenny, etc).

Example:

```
deb http://ceph.newdream.net/gitbuilder-deb-amd64/debian/stable/ maverick main
```

One again needs the public key of the author of ceph for the above auto-build packages as below:

```
$ wget http://ceph.newdream.net/03C3951A.asc  -q -O - | apt-key add -
```

Then,

```
$ apt-get update
```

Do this on all nodes involved in the cluster of ceph (in all MONs, MDSs, and OSDs).

ii **Installing ceph using source code.**

Installing ceph using source code is harder than installing it using the precompiled packages. The precompiled package is, of course, a bit older than the source code, but not more than couple of weeks [1] . In this paper, the first option has been used.

Firstly all ceph dependencies should be installed. Those dependencies are:

```
autotools-dev
autoconf
automake
cdbs
```

```
g++
gcc
git
libatomic-ops-dev
libboost-dev
libcrypto++-dev
libcrypto++
libedit-dev
libexpat1-dev
libfcgi-dev
libfuse-dev
libgoogle-perftools-dev
libgtkmm-2.4-dev
libtool
pkg-config
```

Since they are quite a lot (which is 19), one can install all of them with one command at one time.

```
$ apt-get install debhelper autotools-dev autoconf automake g++ gcc cdbs libfuse-dev
libboost-dev libedit-dev libssl-dev
libtool libexpat1-dev libfcgi-dev libatomic-ops-dev
libgoogle-perftools-dev pkg-config libgtkmm-2.4-dev libcrypto++-dev python-dev
```

The ceph source code can be downloaded from

```
git://ceph.newdream.net/git/ceph.git using the git command:
$ git clone git://ceph.newdream.net/git/ceph.git
```

Then, Be in the ceph directory - which has the source code:

```
$cd ceph
```

To get the stable code (bug fix), instead of using the unstable master code even though it is the most latest, use the following command:

```
$ git checkout b stable origin/stable
```

To build .deb files of ceph:

```
$ dpkg-buildpackage
```

After it finishes building the package, you will see .deb file in the directory where you run the above command (ceph directory). Then use dpkg -i command to install the packages.

```
$ dpkg i <.deb file>
```

For simplicity, the first option is used in this research to install ceph.

4.3 Ceph Cluster Configuration:

Ceph is a distributed and parallel file system which uses a minimum of three nodes to create an abstraction of a single file system called ceph. The main purpose of using cluster of machines is to distribute the work load of huge file system which has a lot of data and storage; and so increases performance and scalability. It also increases the reliability of the file system by striping the data across all OSDs [11] . So; configuring ceph is like configuring many machines for the same purpose or better to call it **Ceph Cluster Configuration.** The configuration should interconnect all involved nodes to communicate in the cluster and make them do their part.

One of the advantages of ceph design approaches is its use of a single central configuration file in order to control all other nodes in the ceph cluster (i.e. all MDSs and OSDs). The option to configure all of them one by one is also available if not tiresome. The central configuration file resides in the Monitor (MON) node located at /etc/ceph/ceph.conf by default. This central configuration has 4 sections or parts to be configured. Those are: Global, MON, MDS, and OSD. All four sections are must to be configured. In general, in this single configuration file you can put all required information about all nodes in the ceph cluster.

Ceph.conf file generally shows the overall composition of cluster of ceph. How many nodes are participating in the cluster and their hostnames, the place where the file system data are stored or path to disks, path to security key rings, which daemons are running, etc. . . can be answered in this single config file. Hence; going to each node and configure is not required except doing small configuration on the 'fetch_config' file to let all other nodes pull down the central configuration file from the MON node.

4.3.1 Configuring ceph.conf (/etc/ceph/ceph.conf)

If ceph is installed successfully (refer Section 4.2), it provides sample.ceph.conf file. Currently the default path to the sample ceph.conf file is /usr/share/doc/ceph/sample.ceph.conf. The location may vary depending on ceph model and/or Linux distribution used; and hence it is better to use 'find' or 'locate' command to look for it if the above path doesn't work:

```
$ find / -name *ceph.conf
$ locate *ceph.conf
```

Then copy this file to /etc/ceph/ceph.conf and configure it according to your interest and setup.

Let's see each section of the central configuration file of ceph (/etc/ceph/ceph.conf):

[Global]

- This portion is mainly concerned about code of communication with the outside world. Ceph supports secure authentication between nodes to make the system more secure which is called **Cephx Authentication** [1].
- Provide keyring directory (optional).

[MON]

- This is the Monitor (MON) node(s) section; in which we provide logging directory (for debugging), the monitor(s) hostname (s) and IP-address (es).
- Provide keyring directory (optional).
- At least one node is required, and at least three for node failures tolerance. The number of MONs should be odd for better result.

[MDS]

- This is the Metadata (MDS) node(s) section where we provide keyring directory (to keep mds's secret encryption keys) and similar to MON portion above we need to provide the hostname(s) and IP-address (es) of MDS(s).
- Provide keyring directory (optional).
- One MDS server is okay. Two is good for redundancy, load-balancing, and for standby; and more according to the size of the file system.

[OSD]

- This is the section of the Object based Storage Device (OSD) nodes. Here, we need to provide device partitions, OSD hostname(s), and IP-addresses.
- Provide keyring directory (optional).
- btrfs file system is recommended for the devices used here. Ceph has an option to automatically mount the provided device by formatting it with btrfs. You don't need to format the partitions in each OSD node. Use 'btrfs devs = /dev/sdx' option.
- As many numbers of OSDs as possible for data replication and better performance.

Ceph has provided one sample configuration file which is located at /usr/share/doc/ceph/sample.ceph.conf by default. The easiest way of configuring ceph.conf is to copy the provided sam-

ple configuration to default location (/etc/ceph/ceph.conf) and modify according to your setup. It is also possible to save the ceph.conf file in some other location different from the default. But, whenever you run ceph command you must provide the path to ceph.conf location which you don't need to do if the default location is used.

(See one sample ceph.conf configuration file used in the project in Appendix C.)

4.3.2 How to join ceph cluster

There are two ways of letting every node in the cluster get or read the central ceph.conf configuration file so as to let them join the ceph cluster.

i **Copy ceph.conf file from MON to all other nodes**

Use 'scp' command to copy the configured /etc/ceph/ceph.conf file from the Monitor node to all other OSDs MDSs, including other MONs if provided. No additional configuration is required in all other nodes.

Example:

```
$ scp /etc/ceph/ceph.conf root@mds1:/etc/ceph/ceph.conf
```

Similarly; do for all nodes.

ii **Use 'fetch_config' script**

Similar to ceph.conf, the default location of fetch_config file is /etc/ceph/fetch_config. It is an executable init script used on each node in the ceph cluster to pull down the central configuration file (/etc/ceph/ceph.conf) from the Monitor node.

Similar to sample.ceph.conf, a sample fetch_config script is also available at: /usr/share/doc/ceph/sample.fetch_config

Or else use again 'find' or 'locate' command to search for it.

```
$ find / -name *fetch_config
```

Copy this file to /etc/ceph/fetch_config and configure it according to each node's position against the Monitor node. It is straight forward (i.e. use scp or cp). Remember that passwordless ssh root access should work (see Section 5.2.1).

sample fetch_config while on OSD1 node

```
1 #!/bin/sh
2 conf="$1"
3
4 ## fetch ceph.conf from some remote location and save it to $conf.
5 ##
6 ## make sure this script is executable (chmod +x fetch_config)
```

```
## examples:
## from a locally accessible file
# cp /path/to/ceph.conf $conf

## from a URL:
# wget -q -O $conf http://somewhere.com/some/ceph.conf

## via scp
# scp -i /path/to/id_dsa user@host:/path/to/ceph.conf $conf

scp I /root/.ssh/ id_rsa root@mon1:/etc/ceph/ceph.conf $conf

```

Then, make /etc/ceph/fetch_config executable.

```
$chmod 755 /etc/ceph/fetch_config
```

Do fetch_config script configuration in all nodes including on the monitor node in which the central configuration file ceph.conf is located. You don't need to run the fetch_config script. It will be used by ceph itself. If fetch_config scipt is provided in all nodes in the ceph cluster, ceph will use the fetch_config script instead of ceph.conf as a means for the nodes to join the cluster.

The first option is of course better unless for some reason the monitor node is far away from any other nodes used in the ceph cluster.

4.3.3 Naming convention in ceph cluster:

Naming restriction is imposed only on OSDs among the nodes which are to be used in the cluster of ceph. It is because of the fact that the OSD ID numbers are used as an index by the daemon for the data structure. So; they should be named in the form of 'osd$id' (where 'id' is integer starting from 0 up to the maximum available OSDs). Ex: osd0, osd1, osd2, osd3 ...

MDSs and MONs are free from the naming rule. They can be named as per the convenience of users.

4.4 Ceph Port numbers and Firewall

Ceph cluster nodes communicate via some port numbers depending on type of daemons. Different daemons use different port numbers in the ceph cluster. See below:

- Monitor (MON) uses cmon daemon and listens on port 6789
- Metadata Server (MDS) uses cmds daemon and listens on any available port

from 6800 to 6803.

- Object-based Storage Device (OSD) uses cosd daemon and listens on any available port from 6800 to 6803.

Accordingly; each nodes firewall should be setup in order to allow those ports coming from your subnet (where ceph cluster is setup) to pass the firewall.

Example:

On Monitor, you need to run the following iptables rule:

```
$ iptables -A INPUT -p tcp -s 10.0.0.0/24 dport  6789 -j ACCEPT
```

For simplicity, you can allow all ports in all nodes using a single command as follows:

```
$ iptables -A INPUT -m multiport -p tcp -s 10.0.0.0/24 --dports 6789,6800:6803 -j ACCEPT
```

4.5 Creating Ceph file system

After ceph installation and configuration are done, we need to create ceph file system with a command called **mkcephfs** (acronym for **m**a**k**e **ceph f**ile **s**ystem). mkcephfs has a short Linux manual page. It is also described in [1].

The general format of the command mkcephfs is [mkcephfs manual page, 4]:

```
mkcephfs [ -a|--allhosts ] [ -c ceph.conf ] [ -k /path/to/keyring.bin ] [ --clobber_old_data ] [ --mkbtrfs ]

Where:-
-a or --allhosts
Initialize all nodes in the cluster via ssh (password-less root access is required)

-c or --conf =/path/to/ceph.conf
Provide the ceph.conf file path if it is different from the default path (/etc/ceph/ceph.conf).

-k /path/to/keyring.bin
Provide path to client admin keyring. The default is /etc/ceph/keyring.bin (default or other path
should be provided explicitly).

--clobber_old_data
        Delete old data in all provided paths both in MON and OSD.
    --mkbtrfs
        Create btrfs file system in each provided OSD devices and mount it.
```

Example of mkcephfs use:

```
$ mkcephfs -c ceph.conf -a --mkbtrfs -k /etc/ceph/keyring.bin
$ mkcephfs -a --mkbtrfs -k keyring.bin
```

(Remember that if -c flag is not invoked, ceph will assume the default location for the ceph.conf file which is in /etc/ceph/ceph.conf) (See sample 'mkcephfs' run in Appendix C).

4.6 Ceph service start/stop

The ceph service daemon controller /etc/init.d/ceph needs to be started after each modification of ceph central configuration file ceph.conf. This init script of ceph makes all ceph daemons in all remote nodes in the ceph cluster to start or stop if -a option is invoked. Password-less ssh root access to all nodes is required by ceph in order to start/stop the daemons in the remote nodes which take part in the ceph cluster.

As it is common to many services, the general usage of ceph daemons star/stop is:

```
$ service ceph [options] (start|stop|forcestop|killall|cleanogs|cleanalllogs) [what]
```

Or,

```
$ /etc/init.d/ceph [options] (start|stop|forcestop|killall|cleanogs|cleanalllogs) [what]
```

Where:-

Options
-v|–verbose
Be verbose
-a|–allhosts
Start/stop all daemons in all nodes in the ceph cluster.
-c foo
Provide ceph.conf file path if it is different from the default (/etc/ceph/ceph.conf)
Commands
start|stop
Start or stop the daemon(s)
forcestop
Force stop (kill -9)
killall
Kill all instance of daemon type
cleanlogs

Clean out log directory

Example:

```
$ service ceph start       (to start all daemons on current node)
$ service ceph -a start       (to start all daemons in all nodes in the ceph cluster)
$ service ceph restart mon       (to restart all monitors on the current node)
$ service ceph stop osd3   (stops osd3 on the current node)
```

Sample ceph service start/stop execution:

Starting ceph service

```
root@addisu-mon1:~# /etc/init.d/ceph -a start
=== mon.0 ===
Starting Ceph mon.0 on mon1...
** WARNING: Ceph is still under heavy development, and is only suitable for **
** testing and review.  Do not trust it with important data.
**
starting mon.0 rank 0 at 10.0.0.8:6789/0 mon\_data /data/mon0
fsid 8f1672f7-6534-f970-3dc3-6fe70999a516

=== mds.mds1 ===
Starting Ceph mds.mds1 on mds1...
 ** WARNING: Ceph is still under heavy development,
  and is only suitable for **
 **          testing and review.  Do not trust it
  with important data.       **
starting mds.mds1 at 0.0.0.0:6800/11645
=== osd.0 ===
Mounting Btrfs on osd1:/data/osd0
Scanning for Btrfs filesystems
Starting Ceph osd.0 on osd1...
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
starting osd0 at 0.0.0.0:6800/4962 osd_data /data/osd0 /data/osd0/journal
=== osd.1 ===
Mounting Btrfs on osd2:/data/osd1
Scanning for Btrfs filesystems
Starting Ceph osd.1 on osd2...
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
starting osd1 at 0.0.0.0:6800/10013 osd_data /data/osd1 /data/osd1/journal
root@addisu-mon1:~# exit
```

Stopping ceph service

```
root@addisu-mds2:~# /etc/init.d/ceph -a stop
=== mon.0 ===
Stopping Ceph mon.0 on mds2...done
=== mds.mds1 ===
Stopping Ceph mds.mds1 on mds1...kill 19447...done
=== osd.0 ===
Stopping Ceph osd.0 on osd1...kill 10243...done
Unmounting Btrfs on osd1:/data/osd0
=== osd.1 ===
Stopping Ceph osd.1 on osd2...kill 21053...done
Unmounting Btrfs on osd2:/data/osd1
```

```
12 root@addisu-mds2:~#
```

4.7 Mounting ceph file system

Mounting ceph file system is possible only if your Linux kernel version is 2.6.34 or later; or else you need to patch your kernel with ceph client sotware which can be found from linux-ceph-client.git tree (refer Section 5.2.1).

Generally, to mount ceph file system, you need to have the monitor (MON) IP address, port number, and a mount point on your client machine. See the format below:

```
$ mount -t ceph monaddr:monport:/ /mountpoint [-v] [-o]
```

Where:
monaddr is the monitor address in the ceph cluster.
monport is the monitor port number (default is 6789)
mountpoint is path to the mount point
-v verbose
-o is to provide name and secret key for secured mount.

If ceph.conf is enabled with authentication cephx, use command cauthtool (acronym for **c**eph **auth**entication **t**ool) to the secret key.

```
$ cauthtool -l /etc/ceph/admin.keyring
```

Sample mount:

```
$ mount -t ceph 10.0.0.1:6789:/ /mnt/point -v -o name=admin, \
secret=AQATSKdNGBnwLhAAnNDKnH65FmVKpXZJVasUeQ==
```

Or

```
$ mount -t ceph 10.0.0.10:/ /mnt/ceph        (with no keyring option; ok for local mounting).
```

You can also get short dpkg package description about ceph and its status on shell command line using dpkg command:

dpkg -s ceph

```
1 root@addisu-osd2:~# dpkg -s ceph
2 Package: ceph
3 Status: install ok installed
4 Priority: optional
```

```
Section: admin
Installed-Size: 14416
Maintainer: Laszlo Boszormenyi (GCS) <gcs@debian.hu>
Architecture: amd64
Version: 0.26-5-g466306d-1maverick
Depends: libc6 (>= 2.6), libcrypto++8, libedit2
(>= 2.5.cvs.20010821-1), libgcc1 (>= 1:4.1.1),
libglib2.0-0 (>= 2.12.0), libglibmm-2.4-1c2a
(>= 2.25.4), libgoogle-perftools0, libgtkmm-2.4-1c2a
(>= 1:2.20.0), libsigc++-2.0-0c2a (>= 2.0.2),
libstdc++6 (>= 4.4.0), hdparm, binutils
Recommends: ceph-client-tools, ceph-fuse,
libceph1, librados2, librbd1, libcrush1, btrfs-tools
Conffiles:
 /etc/logrotate.d/ceph fbf589735803fb8818a27f6a070fd5a4
 /etc/init.d/ceph e8c5dd53443f900624912e59c5bbaabe
Description: distributed storage and file system
Ceph is a distributed storage and network file system designed to provide
excellent performance, reliability, and scalability.
 .
This package contains all server daemons and management tools for creating,
running, and administering a Ceph storage cluster.
Homepage: http://ceph.newdream.net/
root@addisu-osd2:~#
```

4.8 Important ceph commands:

Ceph is not just only a name of the distributed file system, rather it is also used as a control utility for the file system processes and status (see its manual page). It is mainly used to communicate a running ceph distributed file system via the monitor node. Ceph command has three modes of operation: Interactive, watch command line modes. The most common and important ones from each mode are shown below including sample use.

- **Interactive mode:**

  The interactive mode can be started by just running 'ceph' command without any argument. Then, interactively one can use different option to get important information about the status of the ceph file system and about nodes participating in the ceph cluster.

  Examples:

```
$ ceph
ceph>mon stat          (to know the status of monitors in the cluster)
ceph>auth list         (to list the authentication keys of all nodes)
ceph>quit              (or use control+d to exit)
```

  (See sample Interactive mode use in Appendix C).

- **Watch mode:**

This is to see live or a real time status of the file system on the stdout in order to know what is going on currently. It first gives summary of the whole structure of the ceph file system. The summary information are: number of MONs, OSDs, and MDSs joined the cluster, total available disk size to the file system and used disk sizes, up or down nodes, used ports and MON IP-address. After the summary report, it will be ready for live updates as they occur.

If the ceph.conf file is at the default place /etc/ceph/ceph.conf, run the following command without specifying ceph.conf.

```
$ ceph -w
```

If the ceph.conf file is placed different from the default, use c flag to show the direction of ceph.conf file.

```
$ ceph c /path/to/ceph.conf  -w
```

(See sample watch mode use in Appendix C).

- **Command line mode:**

  In this mode; ceph command with some option is used to send an instruction to the monitor node and get result. There are few options to use with ceph command and the most important ones are the two options below.

```
-c        or conf=/path/to/ceph.conf        (if the ceph.conf is not in the default place)
-m monaddress [:port]              (if you  run the command in a remote node)
-s                        (to see an overview of the ceph file system)
```

The common form of ceph command is:

```
$ ceph subsystem command
```

Example:

```
$ ceph auth list              (to list the authentication keys)
$ceph node stat               (to see the status of a node)
```

Checking ceph health

Once in awhile you can check ceph health by executing 'ceph health' command. It gives a summary of the ceph file system current health status. It could be in one of the following three standard health statuses:

- HEALTH_OK,
- HEALTH_WARN, or
- HEALTH_ERR.

Sample execution of 'ceph health' command

```
root@addisu-mon1:~# ceph health
2011-04-22 22:43:05.555800 mon <- [health]
2011-04-22 22:43:05.556158 mon0 -> 'HEALTH_OK' (0)
root@addisu-mon1:~#

root@addisu-mon1:~# ceph health
2011-05-04 00:45:09.067695 mon <- [health]
2011-05-04 00:45:09.068043 mon0 -> 'HEALTH_WARN 1/1/2 osds up/in
Some PGs are: degraded' (0)
root@addisu-mon1:~#
```

(See more sample ceph command line mode use in Appendix C).

The other two important ceph commands are: rados (rados object storage utility), and rbd (manage rados block device (RBD) images). Since they are not used for this particular project, they are discussed here. But, one can read their Linux manual pages.

(See more sample ceph command line mode use in Appendix C).

Recommendations [1]:

- You can improve ceph file system performance by enabling noatime option on all disks.
- All ceph daemons (cmon, cmds, cosd) can be put on the same node If your setup is very small.
- If you want to expand the ceph cluster little more, you can put cmds and cmon on the same node, the cosd daemon on separate node (which is storage node).
- If you again would like to expand it more, you can dedicate separate nodes for each cmds, cmon, and cosd.
- Ceph produces a lot of logs currently. Make sure your log partition is on a fast disk.

Chapter 5

Experiment Setup

Benchmarking ceph distributed file system is to be taken under different conditions and using number of options. The overall hardware specifications and the topology of ceph cluster used in the process of the experiment are stated in this chapter. The experiment is organized according to the type of conditions sets in the benchmarking process. The IOzone options used are discussed in this chapter. The type and number of ceph clients used are also discussed. Finally, three Perl scripts used in the process of benchmarking and data collections are discussed.

5.1 Ceph Topology and Resources used

As it is discussed in the Approach section (Section 5.1); in this project alone, ceph's performance, scalability and reliability is not going to be investigated in its full capacity due to the three main limitations stated in the same section. However, with available resource and time, maximum effort has been exerted to make use of every resource at hand to get better result.

The type and specifications of hardware used to setup and install ceph are listed below and for better look see Table 4.2.

i 3x (Dell PowerEdge R610, 16x Intel(R) Xeon(R) CPU, 24GB, 2x 146GB SATA).

ii 5x (Dell CPU, Intel Core Duo, 4GB, 2x 250GB SATA).

iii 3x (one terabyte extra hard disk for storage)

Figure 5.1 shows the topology of the ceph cluster used in the experiment.

Specification and number of servers used in the research						
Server	**Model**	**CPU (GHz)**	**MEM (GB)**	**DISK(GB)**	**Qty**	**Remark**
Dell PowerEdge	R610	16x2.40	24	6x146	3	Very fast CPU and MEM- recommended for ceph MDSs. 1 is used for 50 VMs
Dell OptiPlex	380	2x2.93	4	2x250	5	1 for MON and; 4 for OSDs..
Extra Disk				1000	3	3 nodes used for OSDs have 1TB extra Disk each

Table 5.1: Resources used in the research.

According to the recommendation of the author of ceph [1] two of the three very powerful servers, the PowerEdge R610, were used for Metadata Servers (MDSs). The other one was used to create 50 virtual machines (VMs) to be used as ceph clients. Out of the rest 5 CPUs (Dell OptiPlex 380), 1 is used for Monitor (MON) and 4 for Object-based Storage Devices (OSDs). In every node used for OSDs an extra one terabyte hard disk has been installed to maximize the storage capacity to scale ceph up to 3TB. According to the recommendation of the author of ceph, the disks in all 4 OSDs are formatted with btrfs file system. In MDSs and MON nodes ext3 was used for local file system.

5.2 Benchmarking ceph

Benchmarking ceph is not as normal as benchmarking traditional local file system. Ceph is a complex system which has many independent components [1]. Each component has its part for the performance and scalability of ceph. In the ceph cluster each component (MDSs, OSDs, and MONs) perform differently. Hence, when we mount ceph and use a benchmark tool to measure its I/O performance, we get the collective performance of all components in the ceph cluster. However, since the main file I/O is processed in the OSD devices, OSDs performance is the very essential factor for the overall performance of ceph file system. The interesting thing is we can separately benchmark each OSD device which is participating in the cluster. The benchmark tool to measure the performance of OSD is not separate software which is needed to be downloaded or installed; rather it is part of the ceph command. See below for how to measure the performance of OSDs.

- **Measuring OSDs performance separately:**

  To measure the OSDs performance one by one, first run the ceph watch mode command on any one of the nodes in the cluster:

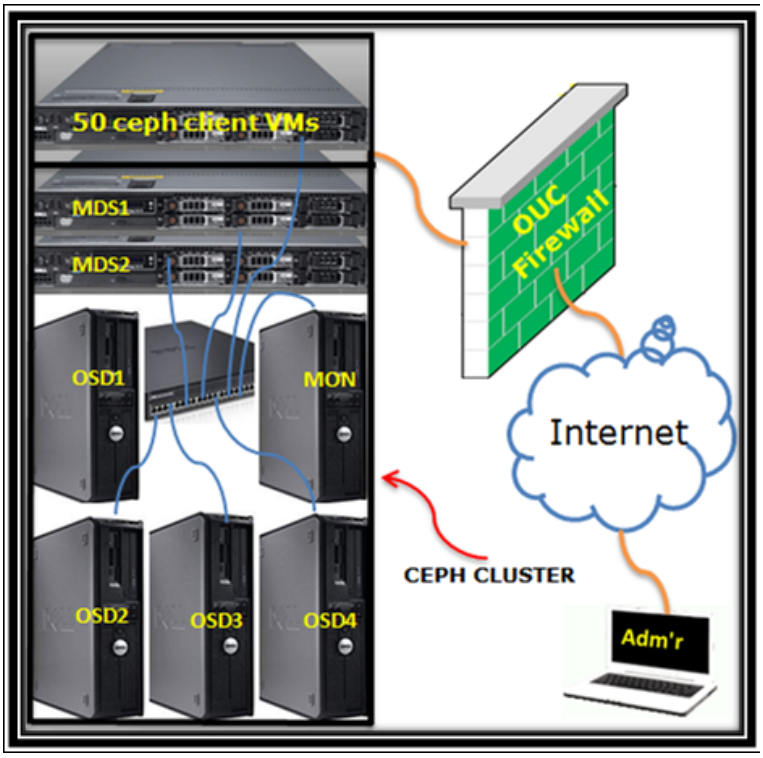


Figure 5.1: Ceph Cluster Topology

```
$ ceph -w
```

Then, run the following command on any other node,

```
$ ceph osd tell # bench       (replace # with  OSD number you want to benchmark)
```

Then, wait and see on the node which you run the ceph wath mode. It will show the average write speed of the particular OSD. See below for sample run.

Sample OSD performance test

```
1
2   root@addisu-mon1:~# ceph osd tell 1 bench
3   2011-04-29 14:15:18.675911 mon <- [osd,tell,1,bench]
4   2011-04-29 14:15:18.676236 mon0 -> 'ok' (0)
5   root@addisu-mon1:~#
6
7   root@addisu-mds2:~# ceph -w
8   2011-04-29 14:14:54.942497    pg v1564: 528 pgs:
9         528 active+clean; 2580 KB data, 2029 MB used,
10        498 GB / 500 GB avail
11  2011-04-29 14:14:54.943174   mds e4: 1/1/1 up
```

```
        0=up:active}
2011-04-29 14:14:54.943193   osd e4: 2 osds:
        2 up, 2 in
2011-04-29 14:14:54.943388   log 2011-04-29
        13:17:43.579509 osd0 10.0.0.1:6800/1730 572
        : [INF] 3.1p0 scrub ok
2011-04-29 14:14:54.943484   mon e1:
 1 mons at {0=10.0.0.8:6789/0}
2011-04-29 14:15:25.517266   log 2011-04-29
        14:17:02.204308 osd1 10.0.0.2:6800/12584 488
        : [INF] bench: wrote 1024 MB in blocks of
        4096 KB in 28.227628 sec at 37147 KB/sec
2011-04-29 14:15:28.524971    pg v1565: 528 pgs:
        528 active+clean; 2580 KB data, 3032 MB used,
    497 GB / 500 GB avail
```

We can see that the average write speed of OSD1 is around 37MB/sec. Similarly we can test other OSDs performance. This is very important since OSDs are the key factor for the overall performance of ceph.

- **Log level**

  Ceph uses around 95% of the total OSDs storage size [1], mainly, due to producing a lot of logs. Since increasing the log level helps for debugging or to identify problems, it is highly recommended to increase the log level. For example, log level 20 is best. However, increasing log level decrease ceph file system performance [1]. The default location of ceph log is /var/log/ceph. See ceph.conf configuration file for how to fix log levels. In this project, log level was minimized to avoid its effect in the ceph performance [1].

5.2.1 Clients used for ceph benchmarking

As ceph is a distributed and parallel file system, it allows hundreds and thousands of clients to access the file system at the same time. As it is mentioned in the Introduction chapter Approach section (section 1.5), testing ceph from hundreds or thousands of clients is beyond the scope of this project due to time and resource limitations. The maximum numbers of clients used are 50. All clients used are kvm based virtual machines (VMs) with ubuntu 10.10 operating system created on a single server. The server which hosts those 50 VMs has 16 processors of 2.4 Ghz each plus 24GB of RAM. It is PowerEdge R610 server; similar to the two servers used for MDSs in the ceph cluster (See table 4.2).

- The VMs specification:

  CPU : 2394 MHz
  cache size : 4096 KB
  MemTotal : 504620 kB

Disk : 2147 MB

The requirement of client machine to mount ceph is described in section . The kernel version of all ceph client VMs is 2.6.35-28-server.

5.2.2 Ceph Cluster LAN Bandwidth and Latency

The LAN network speed and latency is another factor we need to focus on. Since ceph is handling a lot of inputs/outputs in petabyte scale, the LAN where the ceph cluster is setup should be fast enough to cope up with the huge file system I/O operation. The most common problem facing such kind of huge system is bottleneck. We can measure the network speed using 'iperf' command.

Iperf is a very nice tool to deal with network related problems. It uses the idea of client-server model. For example, if you want to test network bandwidth from node1 to node2, simply run 'iperf -s on node2 and 'iperf -c node2_IP' on node1. This is the default use of 'iperf' command which node2 starts listening on TCP port 5001 and iperf will measure the network bandwidth from node1 to node2. It has, of course, many other options. See sample iperf use below:

Sample iperf executions

```
root@addisu-mds2:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.0.0.7 port 5001 connected with 10.0.0.8 port 40002
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.1 sec    114 MBytes  94.2 Mbits/sec

root@addisu-mon1:~# iperf -c mds2
------------------------------------------------------------
Client connecting to mds2, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.8 port 40002 connected with 10.0.0.7 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec    114 MBytes  95.1 Mbits/sec
```

As we can see above the average network bandwidth from mon1 to mds2 is around 95 Mbits/sec.

So, the LAN speed is one of the factors we need to focus while benchmarking ceph. We need to be sure that network bottleneck is not limiting ceph performance.

In this project ceph cluster setup, the LAN network bandwidth has been mea-

sured using 'iperf' command. And, the average bandwidth is 94.5MB/sec.

- **Network Latency:** The network latency can be measured using 'ping' command. The ping command output shows the round-trip time (RTT) for a packet to make a round trip which is starting from client to server and again back to the client machine. Many ping tests against all nodes were taken and the average RTT found is 0.248 milliseconds.

  Note that when looking at the value of network bandwidth and latency, bigger is better for bandwidth, but smaller is better for latency.

5.2.3 Ceph Benchmarking Conditions

The main target of this project is investigating the performance, scalability, and reliability of ceph according to the problem statement stated in chapter 1. In order to fulfill this target, the benchmarking conditions should be designed efficiently so as to be able to answer the three important questions:

- How well ceph performs?
- How scalable ceph is? And,
- How much ceph is reliable?

The ceph file system data I/O read/write speed performance is very important to investigate since it is one part of the problem statement described in Section 1.4. However, since we don't compare it with other similar DFS products under similar conditions, in this project alone, it is hard to know where it stands compare to other DFSs' performance. Hence, in this research, we mainly investigate how ceph performance is affected when file size and number of clients are fluctuating. Hence, ceph scalability is investigated better than ceph performance.

Due to time and resource limitations and due to the nature of the file system which is huge, the file size and number of clients used for the benchmarking test are reduced. And hence, the following three types of benchmarking conditions are set.

- **Type 1: ceph performance and scalability investigation by scaling up number of clients**

  To investigate the effect of fluctuation of number of clients which use ceph file system concurrently, the number of client VMs used in the process of benchmarking scales up from 1 to 50 in multiple of 10 (i.e. 1, 10, ..., 40, 50). For simplicity and to save time, single record size of 256KB and single file size of 1GB are used in all tests of this type. In section 2.2.4 above IOzone Rule 1 says the maximum file size we

use for the test should be bigger than the buffer cache or memory size. That rule is maintained here; because, 1GB is two times bigger than the memory size of each VM which is almost 0.5GB. Client VMs specification is described in section 5.2.1 above.

Script was used to automate the executions of the above 6 cases (1, 10, . . . , 40, and 50) plus 30 iterations of each. Script detail is discussed in the section 5.2.5 below.

– **Type 2: ceph performance and scalability investigation by scaling file size on a single client**

In this case only file size is scaled from 1 to 5GB in multiple of 1 on single client VM in order to investigate the effect of file size fluctuation in the performance of ceph DFS.

– **Type 3: Ceph Reliability investigation by failing one node.**

Simply at the time when 1 MON, 2 MDSs, and all 4 OSDs were joined ceph cluster, ceph reliability was investigated by failing one of the storage nodes while writing was going on. Repeatedly tested and ceph was reliable in all cases, except its total storage size degrades. So; ceph is able to avoid single point of failure.

In all the above benchmarking conditions, the storage capacity of ceph has been increased by expanding ceph cluster depending on the file size used in each client machines while benchmarking ceph and depending on the total number of clients participated in the benchmarking concurrently.

See the next section (section 5.2.4) for the common IOzone command used in all benchmarking tests for condition type 1. Each types of tests has been executed 30 times (iterations=30). See Table 4.3 for better readability of all conditions, and see section 5.2.6 for the Script used to automate the executions and data collections.

5.2.4 Common IOzone command and options used

IOzone is discussed in section 2.2.4. The default or automatic mode uses 13 record sizes from 4KB to 16MB (which is 4, 8, 16 . . . 8192, and 16384 in KB) for each file size test from 64KB to 512MB (which is 64, 128, 256, 512 . . . 262144, 524288 in KB). Again, it measures write, rewrite, read, reread, random read, random write, etc. . . in total 13 different measures; and so 13 outputs for each combination of record size and file size. This is one of the very interesting features of IOzone if one would like to test a file system in many aspects. See Appendix B for sample run and output of default IOzone automatic mode command.

However, due to the scope of this project, most of the above options are omitted for simplicity and to save time. The main target of this project is not to investigate ceph scalability, reliability and performance in its full capacity but to see how much promising it is through different tests with different file sizes and with different number of clients. Although all IOzone options are very interesting and important, all features are not necessary for our purpose. All ceph file system benchmarking has been taken using very few IOzone command options which is common for all tests (except varying file size only).

In the experiment in the fulfillment of Type 1 condition above, one standard or common IOzone command was used in all benchmarking tests which has single file size (1GB) and single record size (256KB) for read/re-read and write/re-write performance of ceph while number of client machines scales up from 1 to 10, and then to 20, ... up to 50, as it is described above. The first common IOzone command used is:

```
$ iozone -i 1 -i 0 -r 256 -s 1g
```

For experiment Type 2 another similar common IOzone command was used on a single client VM by scaling file size from 1GB to 2GB, and then to 3GB, ... up to 5GB. Thanks to IOzone features that we can execute it with the above different file sizes in one command all together, and we will get result of each combination separately. The second IOzone common command used in experiment type 2 is:

```
$ iozone -i 1 -i 0 -r 256 -s 1g -s 2g -s 3g -s 4g -s 5g
```

IOzone output of the above command will be read/re-read and write/re-write speed in Kbytes/sec for file sizes of 1GB, 2GB, ..., 5GB, with constant record size of 256KB. Sample output of the above IOzone command usage is shown in the Result Chapter (section 6.1).

All the above benchmarking conditions are described in Table 4.3 for better readability.

5.2.5 Three Perl Scripts used

In order to automate many benchmarking executions under different conditions and different number of clients, and then to collect results from many nodes Perl scripts are used. It is not practical at all to login in all client nodes (VMs) and run IOzone command one by one, especially in the case of experiment type 1. It is even almost impossible to make 50 client VMs

Ceph Client Side				Ceph Cluster Side				
VMs	IOzone options			Number of nodes joined ceph cluster			Total Disk size (TB)	Iterations
	File size (GB)	Total Size (GB)	Record Size (KB)	MONs	MDSs	OSDs		
Experiment Type 1: scaling number of VMs								
1	1	1	256	1	1	1	0.5	30
10	1	10	256	1	1	1	1	30
20	1	20	256	1	1	2	1.5	30
30	1	30	256	1	2	2	2	30
40	1	40	256	1	2	3	2.5	30
50	1	50	256	1	2	3	3	30
Results of Experiment Type 2: scaling file sizes								
1	1 2 3 4 5	1 2 3 4 5	256	1	2	3	3	30

Table 5.2: Ceph Benchmarking Conditions

run IOzone at the same time manually. The total executions are 4,530 which is (1+10+20+30+40+50)*30 plus 1*30. This case is one simple example to understand the high importance of scripting for system administration work.

Three Perl scripts are used. The first one is used only one time to make the hosting server and all client nodes (i.e. the 50 VMs) ready for the benchmarking experiment. The second script is used to save cron jobs in all VMs going to be used as ceph clients in parallel so as to run the IOzone command concurrently. And, the third Perl script is to collect the IOzone outputs from each VM and organize them in column according to the kind, i.e write, rewrite, read, and reread outputs; by extracting each value from the IOzone output files.

i **The role of the first Perl script**

Simple 'for loop' is used to ssh into all VMs and run the needed commands on each VM as a root (Like: installing ceph and iozone then mount ceph DFS).

For simplicity, the /etc/hosts file of the server is updated with IP to name resolutions for all 50 VMs. Accordingly, the 50 VMs are named: client1, client2, . . . , client49, client50. And, password-less ssh login as a root from the server to all VMs are also allowed. Then, the following

preparations were done on the server and on all client VMs using the first simple Perl script.

i Mount ceph file system on the server and create 50 folders in the root directory so that the 50 clients can use their own folders to run IOzone and save the result. The folders are named: CLIENT_01, CLIENT_02, . . . , CLIENT_49, and CLIENT_50.

ii Install necessary software (i.e. ceph and iozone3) in all client VMs.

iii Create directory for ceph mount point. (Common '/mnt/ceph' directory has been created in all VMs).

iv Mount ceph DFS on all VMs. ($ mount t ceph mon1:/ /mnt/ceph).

ii **The role of the second Perl script**

The second Perl script is more interesting. It enters in all client VMs to be used for benchmarking ceph at a time (or concurrently) and saves small shell script which runs the same common iozone command. It also edits the /etc/crontab file of each VM to save a cron job so as all client VMs to start the benchmarking run at the same time. It takes flags and arguments in the command line to determine number of clients to be used at a time or in parallel and to determine the cron job timing. It, generally, does the following:

- It creates another shell script files for all clients (VMs) to be used for benchmarking ceph concurrently. It writes only two lines in it: the shell interpreter, and then the above common IOzone command. Each newly created shell script redirects the results of the IOzone output to a file and saves it in each VM directory in the ceph file system.
- Then, it writes and saves common cron job with same time setup in /etc/crontab files on all client VMs so as all of them to run the benchmarking concurrently.

The main portion of the script and its usage (flags and arguments options) are shown below.

Flags and arguments options

```
1  sub usage {
2      print "Usage:\n";
3      print "-h Usage\n";
4      print "-v Verbose\n";
5      print "-d Debug\n";
6      print "-n Number of VMs\n";
7      print "-m cron minute\n";
8      print "-H cron hour\n";
9      print "-D cron date\n";
10     print "-M cron month\n";
```

```
    print "-w cron week\n";
}
```

Main portion of the second script

```
for($i=1; $i<=$vms;$i++)
{
system("touch /root/SHELL/cephIO_for_$vms\VMs_onVM$i.sh");
system("echo \'#!/bin/bash\' > /root/SHELL/cephIO_for_$vms\VMs_onVM$i.sh");
system("echo \'iozone -i 1 -i 0 -r 256 -s 1g > \ cephIO_for_$vms\VMs_onVM$i.txt\'
>>/root/SHELL/cephIO_for_$vms\VMs_onVM$i.sh");
system("scp /root/SHELL/cephIO_for_$vms\VMs_onVM$i.sh \
root\@client$i:/root/cephIO_for_$vms\VMs_onVM$i.sh");
system("ssh root\@client$i \"chmod 755 /root/cephIO_for_$vms\VMs_onVM$i.sh\"");
system("ssh root\@client$i \"echo \'$min $hr $date $mon
$wk root (cd \ /mnt/ceph/CLIENT_$i

&& /root/cephIO_for_$vms\VMs_onVM$i.sh)\'
>>/etc/crontab\"");
}
```

Sample Execution of the 2nd script when 50VMs are used

```
./ceph_iozone_general.pl -n 50 -i 2 -m 35 -H 18 -D 14 -M 5 -w 6
```

According to the above execution, cron jobs will be saved in all 50 VMs in order to execute the shell script located in each VM root home directory which is saved by the above same Perl script (ceph_iozone_general.pl). The 50 VMs run their own shell scripts at the same time on May 14 at 18:35 according to the above particular execution.

iii The role of the third Perl script

The third Perl script is also interesting. It enters in all VMs folders in the ceph file system... opens the IOzone output file and extract the necessary value.... and then opens a new file to save the outputs of write/rewrite and read/reread speed values in the file organizing by columns. Similar to the second script it also receives flags and arguments in the command line to determine the number of VMs from which the outputs are going to be collected.

The main portion of the script and its usage (flags and arguments options) are shown below.

Flags and arguments options

```

sub usage {
    print "Usage:\n";
    print "-h Usage\n";
    print "-v Verbose\n";
    print "-d Debug\n";
    print "-n Number of VMs\n";
    print "-r output order\n";
}
```

Main portion of the third script

```
 for($i=1; $i<=$n; $i++)
{
    open(file, "/mnt/ceph/CLIENT_$i/cephIO_for_$n\VMs_onVM$i.txt$r") \
 or die "Error: $!\n";
    while($line = <file>){
        chomp $line;
        if ($line =~ /\s+1048576\s+\d+\s+\d+.*/){
            @array = split /\s+/, $line;
            push @write, $array[3];
            push @rewrite, $array[4];
            push @read, $array[5];
            push @reread, $array[6];
        }
    }
}

open(file2,">/root/cephIO_output/$n\VMs/$r/cephIO_$n\VMs_output_order_$r.txt");
print file2 "write\trewrite\tread\treread\n";

my $m = $n-1;
for($i=0; $i<=$m; $i++)
{
    print file2 "$write[$i]\t$rewrite[$i]\t$read[$i]\t$reread[$i]\n";
}
```

The complete lists of the above three Perl scripts are found in Appendix D.

Chapter 6

Ceph Benchmark Results

Ceph performance, scalability, and reliability have been measured and investigated according to the conditions discussed in Chapter 5 (Experiment Setup). A lot of benchmarking tests under different conditions were done and results have been collected. This chapter mainly concentrates only on the results found and typical sample executions and outputs from each condition types. Results are outputs of IOzone commands used in benchmarking ceph DFS under three major conditions. The IOzone '-i' option is used to limit the type of outputs to be the write/re-write and read/re-read speeds in Kbytes/sec of ceph DFS.

Ceph DFS benchmarking was done from different number of client machines at a time (or concurrently) in order to investigate ceph performance and scalability. That is, of course, one of the three conditions set in the experiment setup (which is by scaling up the total number of client machines used to mount ceph and run IOzone concurrently with same file size). The second main condition is scaling up the file size used in each benchmarking by being on single client only. And, the third condition is investigating the reliability of ceph by failing one of the OSD nodes deliberately.

Every benchmarking tests are iterated 30 times so as to evaluate and interpret the results (sample outputs) as correctly as possible using statistical calculations (like: average or mean, outliers, standard deviation, max, min, median, confidence interval, standard error of mean, etc of the raw outputs). 'R' software for its interesting self explanatory boxplot and MS Excel for its simplicity and to show patterns of raw data are used for the statistical calculations and graphs of the sample outputs of each benchmarking condition. The meaning of each statistical terms used are listed in table **??**, and the meaning of each boxplot value of R is shown in figure 6.1.

This chapter is organized according to the above three main benchmarking

Terms	Meaning
Min	Sample's Minimum value
Max	Sample's Maximum value
Mean	Sample's average value
Median	Sample's middle value after listing in ascending or descending order
STDEV	Sample's Standard Deviation value $= 1\frac{}{N-1\times\sqrt{\sum(S-M)^2}}(6.1)$, Where , S = sample value, N is sample's size, and M is Sample's Mean value.This value shows in how extent the samples are scattered or it just tells us how much each sample value is far from the Sample's Mean. So; obviously, lesser STDEV is better, as it shows the samples are less scattered.
SEM	Standard Error of the Mean $= 1\frac{}{\sqrt{N}\times STDEV}(6.2)$ It tells us the precision of the sample's Mean (how far the sample's Mean is from the true Mean). It is always less than STDEV. As the sample size (N) increases, SEM will be much smaller than STDEV.
CONF	95% CONFIDENCE INTERVAL of the given samples using excel [CONFIDENCE (0.05, **STDEV**, sample size)]. It can be interpreted as with 95% confidence the true mean lies in between **(M CONF) & (M + CONF),** or **[(M CONF) < TRUE MEAN < (M + CONF)].** So; lesser CONF value is better, as it shows the sample's Mean value is closer to the true mean value.

Table 6.1: Meaning of statistical terms used in the research.

conditions for better readability. One sample execution and output are also shown for each case.

6.1 Results of Experiment Type 1: Scaling number of clients

Under this condition there are 6 sub-conditions which are the six different number of client machines (VMs) used to benchmark ceph by running iozone command concurrently (i.e. when 1, 10, . . . , 40, and 50 VMs are used). The Perl script discussed in Chapter 5 plays a great role here. Benchmarking ceph using single client node could be easy, but benchmarking it from plenty of client machines at the same time and collecting the results from all nodes is not simple without the help of script.

So; in this sub section we will see sample executions and outputs found when 1VM, 10VMs, . . . , 40VMs, and 50VMs were used as ceph clients performing the benchmarking work concurrently. Monitoring tools used to monitor the process from client and server side are also discussed in this

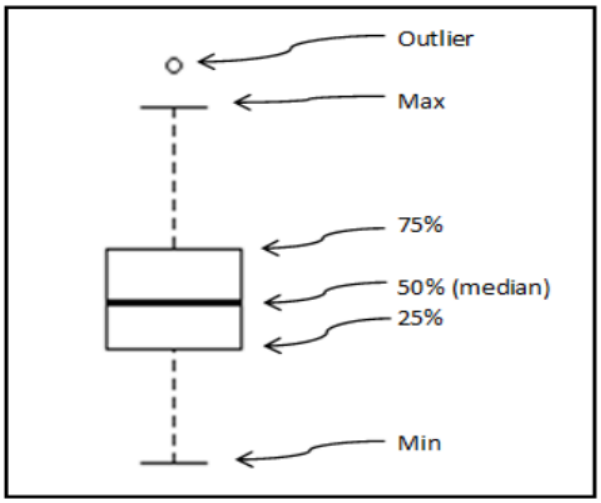


Figure 6.1: Meaning of boxplot values of software 'R'.

section. R boxplots and Excel graphs are used to show the overall results.

– **Summary of condition 1 experiment setup:**

  * Both Ceph cluster and clients are in the same LAN
  * The 50 clients are all VMs created on single server
  * 6 sub-conditions: when 1VM, 10VMs, . . . , 50VMs participate in the benchmarking at the same time.
  * Common IOzone command and options used at each benchmarking execution:

```
$ iozone -i 1 -i 0 -r 256 -s 1g
```

  Section 6.1.1 shows some typical sample executions and outputs, section 6.1.2 talks about how the benchmarking process was monitored from both ceph client and server side, and section 6.1.3 shows Excel graphs and boxplots of the results of all 6 sub-conditions under this condition.

6.1.1 Experiment Type 1 Sample Executions and Outputs:

i **Sample second Perl script execution to save cron jobs and IOzone output:**

The three Perl scripts used in the experiments are discussed in the Ex-

periment Setup chapter in section 5.2.6. Just one typical execution and sample output are shown below for the two main scripts (the second and third scripts) used in the process of benchmarking

Sample Execution of the second script to save cron jobs in 50VMs

```
root@ceph-client:~# ./ceph_iozone_general.pl -n 50 -i 2 -m 35 -H 18 -D 14 -M 5 -w 6
```

Sample output of IOzone on VM48 when 50 VMs used concurrently

```
root@ceph-client48:~# cat /mnt/ceph/CLIENT_48/cephIO_for_50VMs_onVM48.txt2
    Iozone: Performance Test of File I/O
        Version $Revision: 3.308 $
        Compiled for 64 bit mode.
        Build: linux

    Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
            Al Slater, Scott Rhine, Mike Wisner, Ken Goss
            Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
            Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
            Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy,
            Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root.

    Run began: Sun May 15 01:25:01 2011

    Record Size 256 KB
    File size set to 1048576 KB
    Command line used: iozone -i 1 -i 0 -r 256 -s 1g
    Output is in Kbytes/sec
    Time Resolution = 0.000001 seconds.
    Processor cache size set to 1024 Kbytes.
    Processor cache line size set to 32 bytes.
    File stride size set to 17 * record siz
    random  random    bkwd   record   stride KB  reclen   write

    rewrite    read    reread      write    read  rewrite
    1048576     256     323     354     350     287

iozone test complete.
root@ceph-client48:~#
```

ii **Sample third Perl script execution for data collection:**

To extract and collect the write/rewrite and read/reread outputs from all VMs and organize them together in a column is done by the third script. It also saves the organized values in a file. The sample execution below shows when the second iteration of IOzone output for 50 VMs is being collected.

```
root@ceph-client:~/cephIO_Perl# ./arrang_output.pl -n 50 -r 2
write   rewrite read    reread
258     268     225     249
262     276     242     237
265     266     233     241
241     268     232     246
253     258     235     244
```

```
268   273   236   238
253   278   235   247
248   268   232   246
.
.
. (total of 50 lines from the 50VMs)
```

6.1.2 Monitoring the Benchmarking Process.

Both ceph cluster and ceph client sides were monitored while the process of benchmarking run was ongoing.

– **Monitoring from Ceph Cluster Side**

The whole ceph cluster status while client machines are writing some data on it could be monitored using ceph command tools to follow what is going on the ceph DFS; especially, to see the storage usage and the OSD nodes status. Monitoring the process is very important to control the system. For example: any one of the nodes in the ceph cluster may fail at any time, and the cluster may need to be expanded because of high demand from clients. Thanks to the advanced architectural design of ceph that both of them could easily be fixed without the need of stopping ceph, but it should be monitored (see the methodology chapter section . for how to do that). One also gets other important information, summary of the whole status of ceph, via ceph monitoring tools.

The most important ceph monitoring tools are ceph watch mode command 'ceph -w' and ceph status command 'ceph -s'. 'ceph -w' command first gives the same output as 'ceph -s' which is summary of the status of ceph including how many MONs, OSDs, and MDSs are being used, total storage it currently has (total and used separately), and then keeps it open (live) to see the change of status every second if any. ceph health could also be checked once in awhile by executing 'ceph health' command. All of them are, actually, discussed in the methodology chapter (chapter 4).

See below the sample usage of 'ceph -s' and 'ceph -w' commands when the ceph cluster consisted of 1 MON, 3 OSDs, and 2 MDSs; and when 50 VMs are used in the benchmarking process concurrently.

Sample ceph -s ceph -w and ceph health usage

```

root@addisu-mon1:~# ceph -s
2011-05-09 18:34:01.338738    pg v531: 792 pgs: 792 active+clean; 2049 MB data,
 7141 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:01.340590   mds e5: 1/1/1 up {0=up:active}, 1 up:standby
2011-05-09 18:34:01.340629   osd e5: 3 osds: 3 up, 3 in
2011-05-09 18:34:01.340688   log 2011-05-09 17:53:02.478740 osd0
```

```
10.0.0.1:6800/7457 277 : [INF] 3.1p0 scrub ok
2011-05-09 18:34:01.340780   mon e1: 1 mons at {0=10.0.0.8:6789/0}
root@addisu-mon1:~#

root@addisu-mon1:~# ceph w
2011-05-09 18:34:01.338738    pg v531: 792 pgs: 792 active+clean; 2049 MB data,
 7141 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:01.340590   mds e5: 1/1/1 up {0=up:active}, 1 up:standby
2011-05-09 18:34:01.340629   osd e5: 3 osds: 3 up, 3 in
2011-05-09 18:34:01.340688   log 2011-05-09 17:53:02.478740 osd0
10.0.0.1:6800/7457 277 : [INF] 3.1p0 scrub ok
2011-05-09 18:34:01.340780   mon e1: 1 mons at {0=10.0.0.8:6789/0}
2011-05-09 18:34:01.918564    pg v532: 792 pgs: 792 active+clean; 2049 MB data,
 7193 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:05.867884    pg v533: 792 pgs: 792 active+clean; 2049 MB data,
 7237 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:06.899830    pg v534: 792 pgs: 792 active+clean; 2049 MB data,
 7298 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:10.871026    pg v535: 792 pgs: 792 active+clean; 2049 MB data,
 7340 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:11.915810    pg v536: 792 pgs: 792 active+clean; 2049 MB data,
 7348 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:15.877005    pg v537: 792 pgs: 792 active+clean; 2049 MB data,
 7350 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:16.911737    pg v538: 792 pgs: 792 active+clean; 2049 MB data,
 7350 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:20.891111    pg v539: 792 pgs: 792 active+clean; 2049 MB data,
 7362 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:21.926345    pg v540: 792 pgs: 792 active+clean; 2049 MB data,
 7366 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:25.880096    pg v541: 792 pgs: 792 active+clean; 2049 MB data,
 7366 MB used, 2776 GB / 2790 GB avail
2011-05-09 18:34:26.914522    pg v542: 792 pgs: 792 active+clean; 2049 MB data,
 7358 MB used, 2776 GB / 2790 GB avail
^C
root@addisu-mon1:~#

root@addisu-mon1:~# ceph health
2011-05-15 04:38:43.206766 mon <- [health]
2011-05-15 04:38:43.207149 mon0 -> 'HEALTH_OK' (0)
root@addisu-mon1:~#

```

– **Monitoring from Ceph Client side**

IOzone command is the main processes to be monitored from the ceph client side (its CPU, RAM, and network usage). There are, of course, different kinds of monitoring tools to monitor CPU, and RAM used by a process, and network status at any interface. Some of the tools are 'top', 'ps', 'mpstat','iostat' 'netstat', 'ifconfig', 'iftop', 'du', 'df', etc…

'ps' , 'top' and 'iftop' are the most commonly used as a client side monitoring tools in this research. 'ps' and 'top' are used to monitor the CPU and MEM usage of IOzone command and 'iftop' to monitor the amount of data passing through the network every second. 'df' and

'du' commands are also used to see the storage status of the mounted ceph file system. See sample usages below:

Sample usage of 'top'

```
root@ceph-client5:~# ps aux | grep iozone
root       958  0.3  3.4  47092 17416 ?        S    20:55 \
 0:08 iozone -i 1 -i 0 -r 256 -s 1g
root      1092  0.0  0.1   8952   856 pts/1    S+   21:42 \
 0:00 grep --color=auto iozone
root@ceph-client5:~# top -p 958
top - 21:42:35 up  2:01,  1 user,  load average: 0.10, 0.14, 0.09
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.2%sy,  0.0%ni, 24.9%id, 74.3%wa,  0.0%hi, \
0.5%si,  0.0%st
Mem:    504620k total,   457348k used,    47272k free, \
  14828k buffers
Swap:   153596k total,        0k used,   153596k free,  \
 384996k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  958 root      20   0 47092  17m  592 S  0.0  3.5   0:08.58 iozone

root@ceph-client40:~# df -hT /mnt/ceph
Filesystem    Type    Size  Used Avail Use% Mounted on
10.0.0.8:/    ceph    2.8T   67G  2.7T   3% /mnt/ceph
root@ceph-client40:~#
```

'iftop' command is very interesting which displays live bandwidth usage of each node connected to the given interface. For example; while being on the server hosting the 50 VMs used as ceph clients we can run 'iftop' command and observe the bandwidth usage of each ceph cluster node that are connected to the client server. We can easily understand the advantage of decoupling metadata and actual data in the distributed file system. 'iftop' shows that biggest data sizes are moving directly to storage nodes (OSDs). And, smaller sizes to metadata nodes (MDSs), then smallest data sizes to Monitor node (MON) of the ceph cluster and vice versa; just according to their purpose in the ceph file system. See Figure 6.2, and 6.3 for the illustration. Figure 6.3 shows when the benchmarking run is completing and hence metadata and monitor are left to close the session as the actual data movement to and from the three OSD nodes are done.

6.1.3 Graphical Representation of Results of Experiment type 1.

The R program boxplot is almost self explanatory. Without the need of calculating the most important statistical values of a raw data, it shows

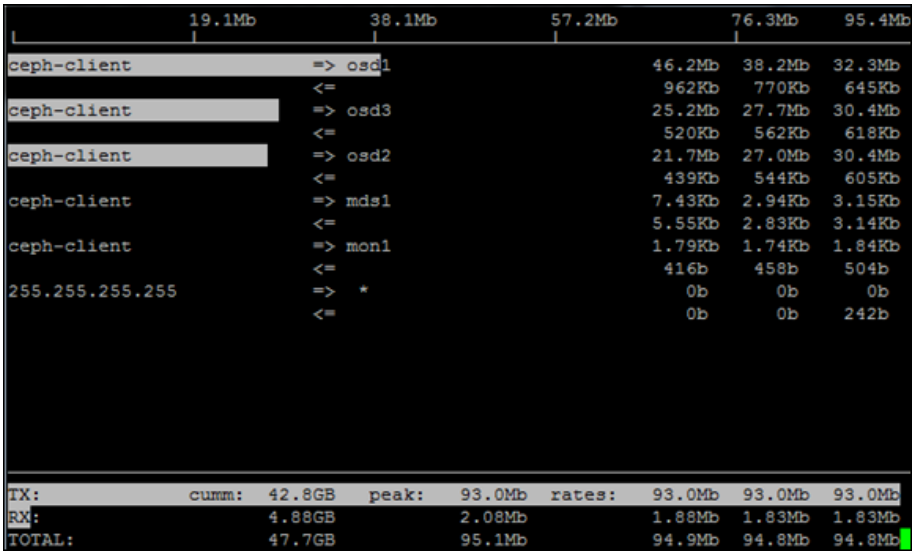


Figure 6.2: Output of 'iftop' command while iozone command was running from 10VMs.

where the 25, 50, and 75 percent quartiles of the values are concentrated plus the data min, max, and outliers if any. Outliers are values in the data which are very distant from the rest of the data. Figure 6.1 shows meaning of boxplot values. Boxplot is also very helpful tool to compare two or more similar data. By just observing where the boxes lies in the graph one easily compare in different aspects. In this chapter boxplots are used mainly to compare the write/rewrite and read/reread speed of ceph DFS when 1VM, 10VMs, 20VMs, 30VMs, 40VMs, and 50VMs are used as a client to benchmark it using IOzone concurrently.

Excel graphs (line and bar graphs) are also used to show the pattern of the raw data values. And, it is also used to show and compare average values of each sub-condition.

Huge value variation have been observed between results when 1VM and the rest of 5 different number of VMs (10, 20, . . . , 50VMs) are used as ceph clients; which, as a result, creates questions about the scalability of ceph which will be discussed in the next chapter (Data Analysis chapter). For better visibility and readability each type of graph (line graphs, bar graphs, and boxplots) are divided into two: when the number of clients (VMs) scales from 1 to 50, and when it scales from 10 to 50 excluding the results found from 1VM.

When trying to compare the read/write speed of 1VM, 10VMs, . . . , 50VMs using boxplots in one graph, the boxplots are not clearly visible because of the values gap between the different number of clients

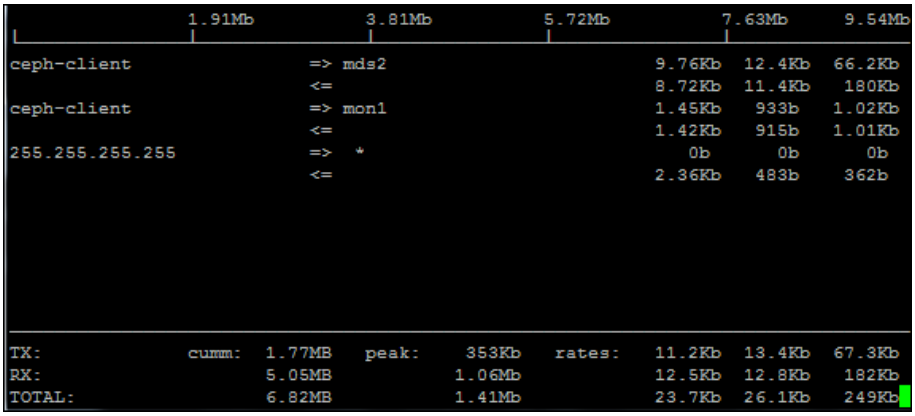


Figure 6.3: : Output of 'iftop' command when IOzone command is closing its session with Ceph DFS.

(VMs) used. Hence; for better visibility and readability of boxplots, collection of write/read speed for 1, 10, 30, and 50VMs are also shown in Figure 6.12 and 6.13.

The average read/re-read and write/rewrite speed of each sub-condition are calculated using MS Excel and shown in one graph except showing one more by excluding 1VM from the table for better readability (See Figures 6.14 and 6.15).

The standard Error of Mean (SEM) which is = STDEV/SQRT(data size), and Confidence intervals (CONF) of each type of data are also calculated and shown in bar graphs (Figure 6.16 and 6.17) below for one to easily get an idea of the accuracy of the data. To reduce the numbers of graphs which talks about similar cases, only the read and write speed values are considered for illustration. For anyone who is interested to see the rest of the values, they are found in Appendix E.

The analysis of the implications and meaning of the values and graphs shown in this chapter are discussed in the next chapter (Chapter 7).

6.2 Results of Experiment Type 2: Scaling file sizes

In this condition only single client node is used at a time by scaling the file size option of IOzone command from 1 to 5GB in multiple of 1. The execution and outputs are almost similar to experiment type 1. And, similar to the first experiment, the record size used is 256KB in all cases. This is to investigate ceph performance and scalability in

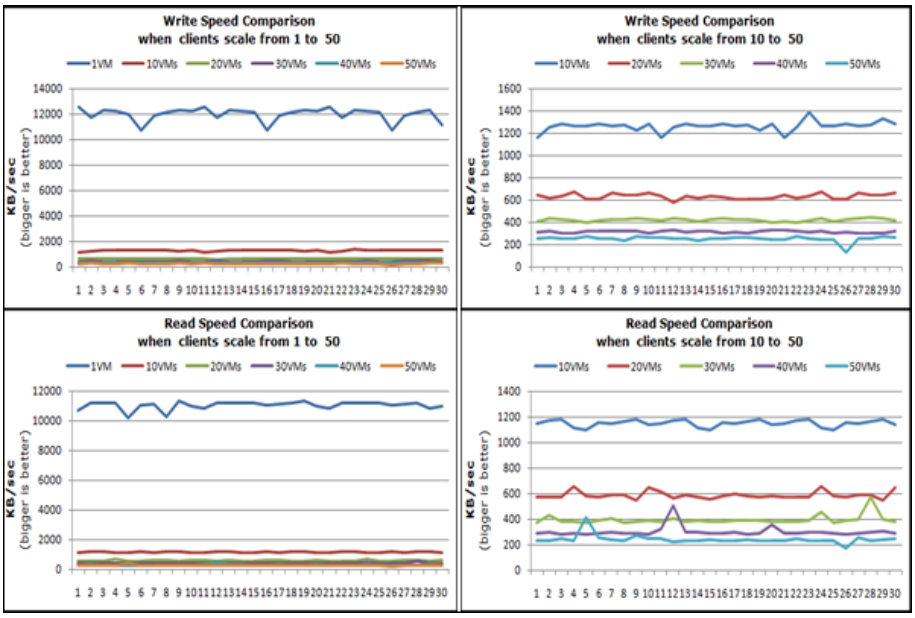


Figure 6.4: Line graph of write and read performance comparison when number of clients scale up (including and excluding 1VM result).

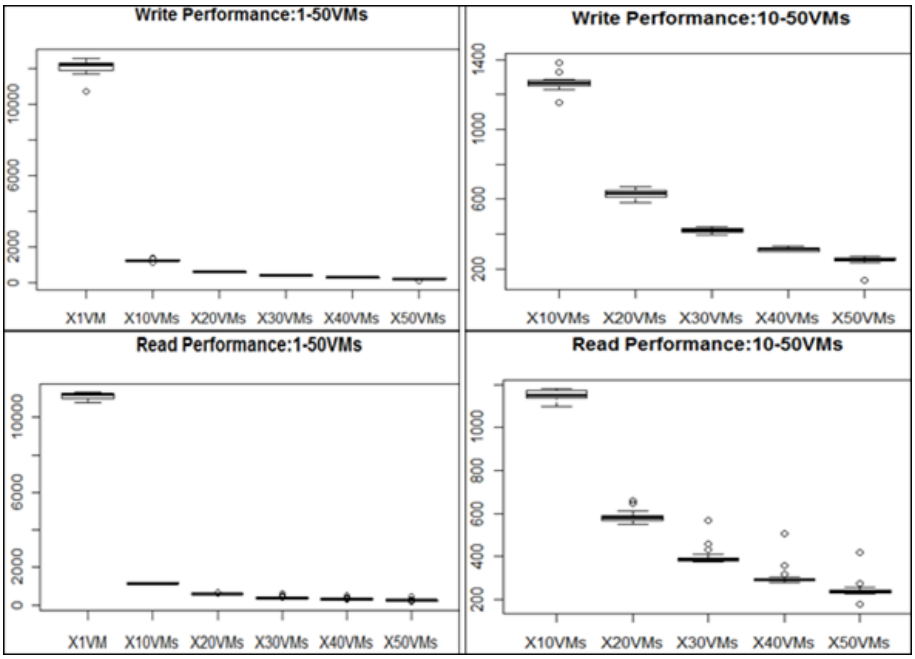


Figure 6.5: Boxplots of write and read performance comparison when number of clients scale up (including and excluding 1VM result).

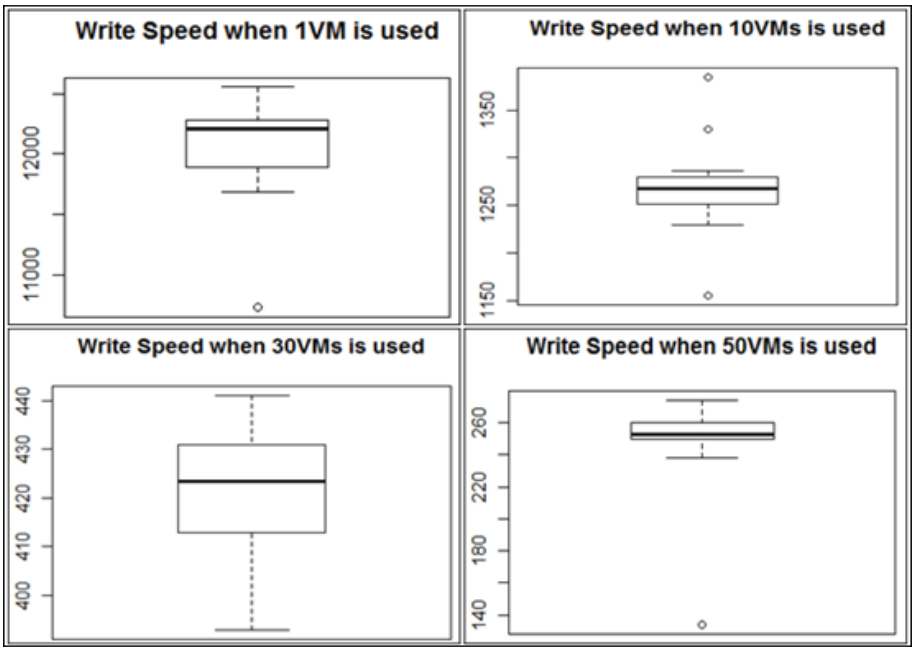


Figure 6.6: Boxplots of write performance when 1VM, 10VMs, 30VMs, and 50VMs are used as ceph clients.

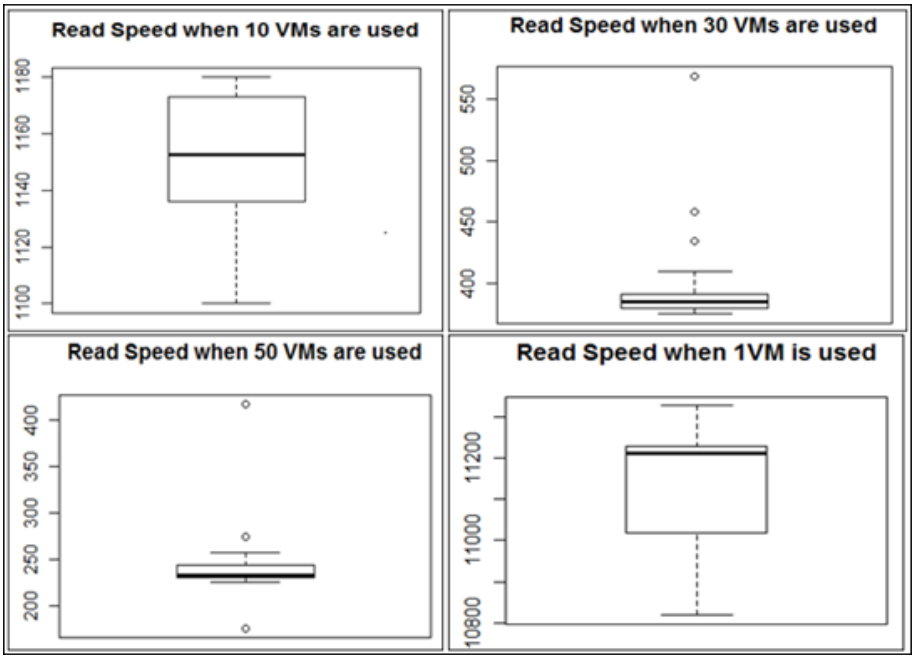


Figure 6.7: Boxplots of read performance when 1VM, 10VMs, 30VMs, and 50VMs are used as ceph clients.

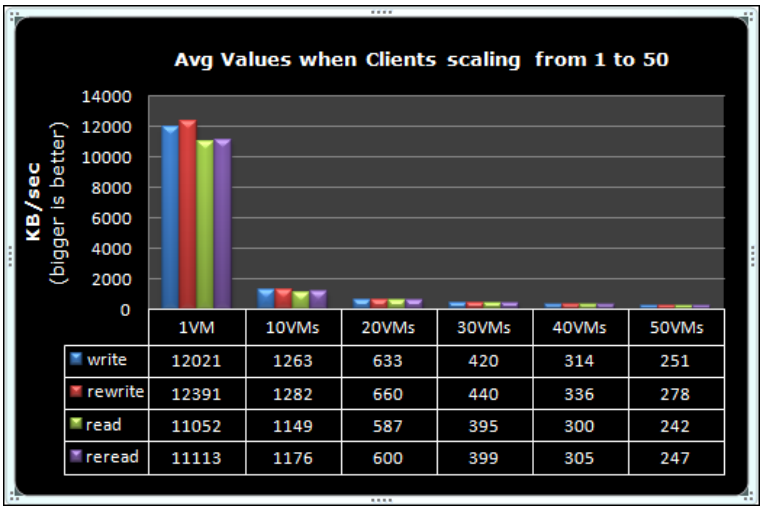


Figure 6.8: Average write/rewrite and read/reread speed when ceph clients scale from 1 to 50.

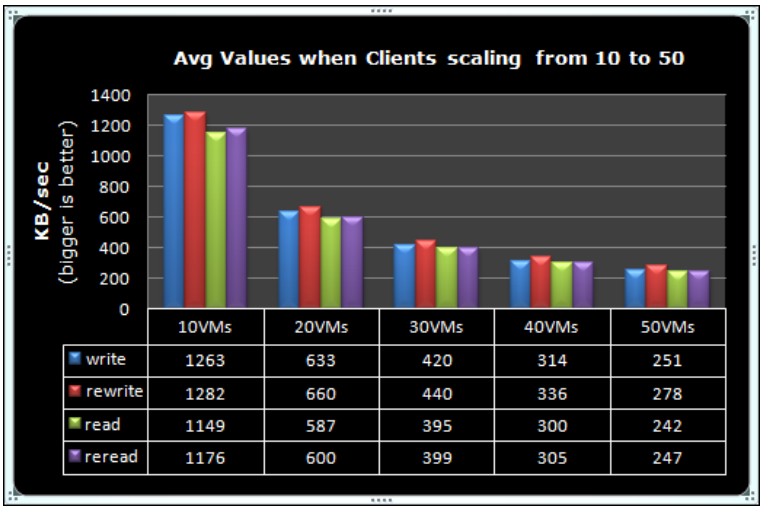
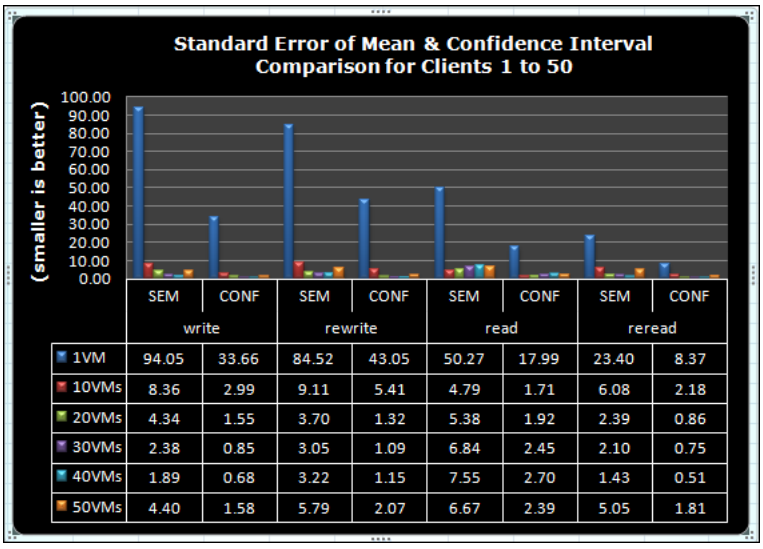
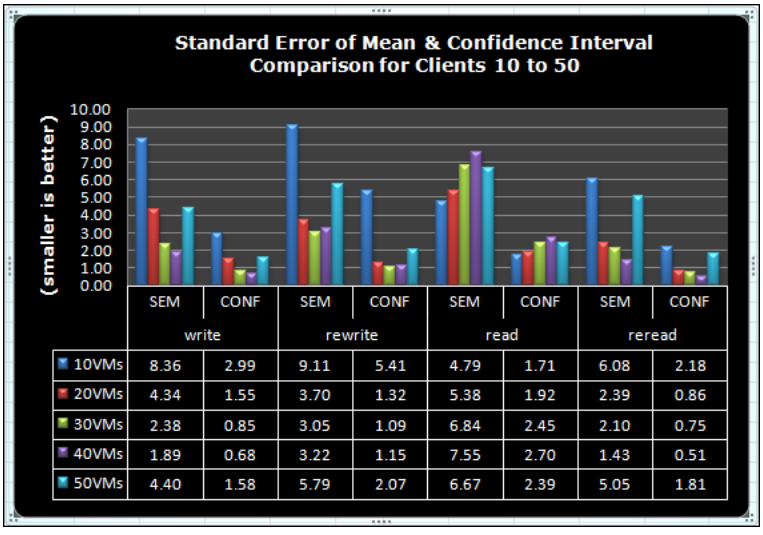


Figure 6.9: Average write/rewrite and read/reread speed when ceph clients scale from 10 to 50.



	write		rewrite		read		reread	
	SEM	CONF	SEM	CONF	SEM	CONF	SEM	CONF
1VM	94.05	33.66	84.52	43.05	50.27	17.99	23.40	8.37
10VMs	8.36	2.99	9.11	5.41	4.79	1.71	6.08	2.18
20VMs	4.34	1.55	3.70	1.32	5.38	1.92	2.39	0.86
30VMs	2.38	0.85	3.05	1.09	6.84	2.45	2.10	0.75
40VMs	1.89	0.68	3.22	1.15	7.55	2.70	1.43	0.51
50VMs	4.40	1.58	5.79	2.07	6.67	2.39	5.05	1.81

Figure 6.10: SEM and CONF of raw data when clients scale from 1 to 50.



	write		rewrite		read		reread	
	SEM	CONF	SEM	CONF	SEM	CONF	SEM	CONF
10VMs	8.36	2.99	9.11	5.41	4.79	1.71	6.08	2.18
20VMs	4.34	1.55	3.70	1.32	5.38	1.92	2.39	0.86
30VMs	2.38	0.85	3.05	1.09	6.84	2.45	2.10	0.75
40VMs	1.89	0.68	3.22	1.15	7.55	2.70	1.43	0.51
50VMs	4.40	1.58	5.79	2.07	6.67	2.39	5.05	1.81

Figure 6.11: SEM and CONF of raw data when clients scale from 10 to 50.

the situation when the amount of file or data used by a client machine scales up.

Sample IOzone output of Experiment type 2

```
root@ceph-client:~/CEPH_OUTPUT/1to5# cat cephIO_1vm_1-5g_onVM10
        Iozone: Performance Test of File I/O
                Version $Revision: 3.308 $
                Compiled for 64 bit mode.
                Build: linux

        Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
                     Al Slater, Scott Rhine, Mike Wisner, Ken Goss
                     Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
                     Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
                     Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy,
                     Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root.

        Run began: Mon May  9 18:24:10 2011

        Record Size 256 KB
        File size set to 1048576 KB
        File size set to 2097152 KB
        File size set to 3145728 KB
        File size set to 4194304 KB
        File size set to 5242880 KB
        Command line used: iozone -i 1 -i 0 -r 256 -s 1g -s 2g -s 3g -s 4g -s 5g
        Output is in Kbytes/sec
        Time Resolution = 0.000001 seconds.
        Processor cache size set to 1024 Kbytes.
        Processor cache line size set to 32 bytes.
        File stride size set to 17 * record size.
                                                    random  random
            KB  reclen   write rewrite    read    reread    read   write
       1048576     256   11692   12370   11211   11214
       2097152     256   11844   11746   11046   11113
       3145728     256   11459   11589   10750   10888
       4194304     256   11491   11478   10658   10673
       5242880     256   11542   11532   10661   10658

iozone test complete.
root@ceph-client:~/CEPH_OUTPUT/1to5#
```

6.2.1 Graphical Representation of Results of Experiment type 2

Similar to above in the first experiment, here also boxplots and MS Excel line and bar graphs are used to show the results found in this experiment. For better visibility and simplicity, write and read performance are selected for the graphs. They are organized in a way for one to get complete information about the results of all cases and for easy comparison. The average values are also shown to compare the effect of scaling up file size in the performance of ceph file system.

We can clearly observe from the graphs that when file size scales up, both read and write performance degrades. Their average values, especially, show the performance degradation when file size increases (see figure 6.24). In the next chapter, data analysis is discussed in detail.

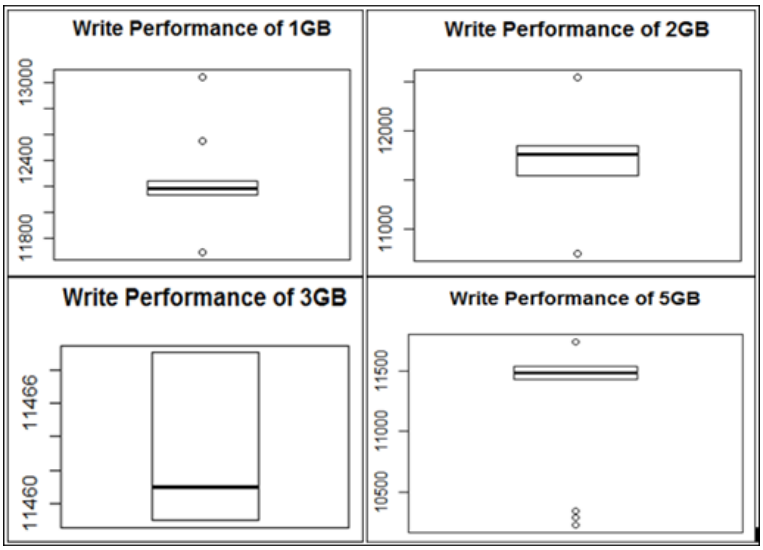


Figure 6.12: Write performance boxplots of 1GB, 2GB, 3GB, and 5GB file size

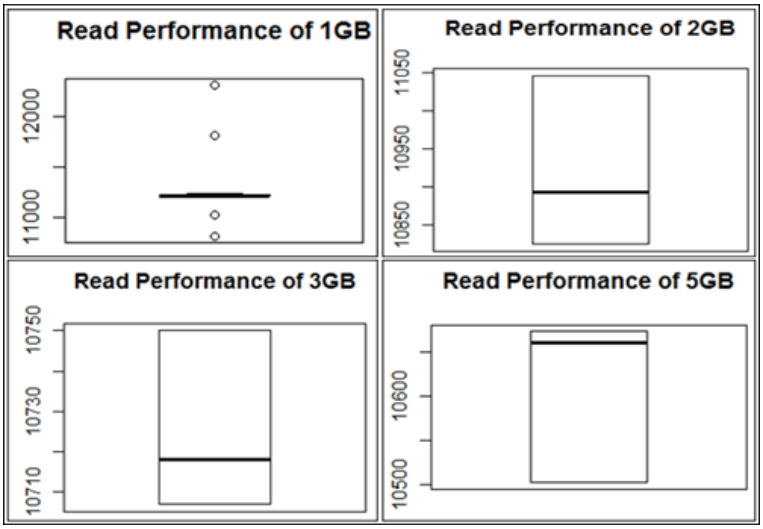


Figure 6.13: Read performance boxplots of 1GB, 2GB, 3GB, and 5GB file size

6.3 Result of Experiment type 3: Reliability check

The reliability of ceph DFS was tested several times by failing one of the storage nodes (OSDs) turn by turn at the time when three OSDs were used. In all cases, it didn't collapse, except the total storage capacity degraded because of the storage node failure. There was no data loss in the time of single OSD node failure. In ceph architecture node failure is a norm rather than exception. This assertion of ceph has been proved in this project. It proves its reliability strength and hence avoids single point of failure.

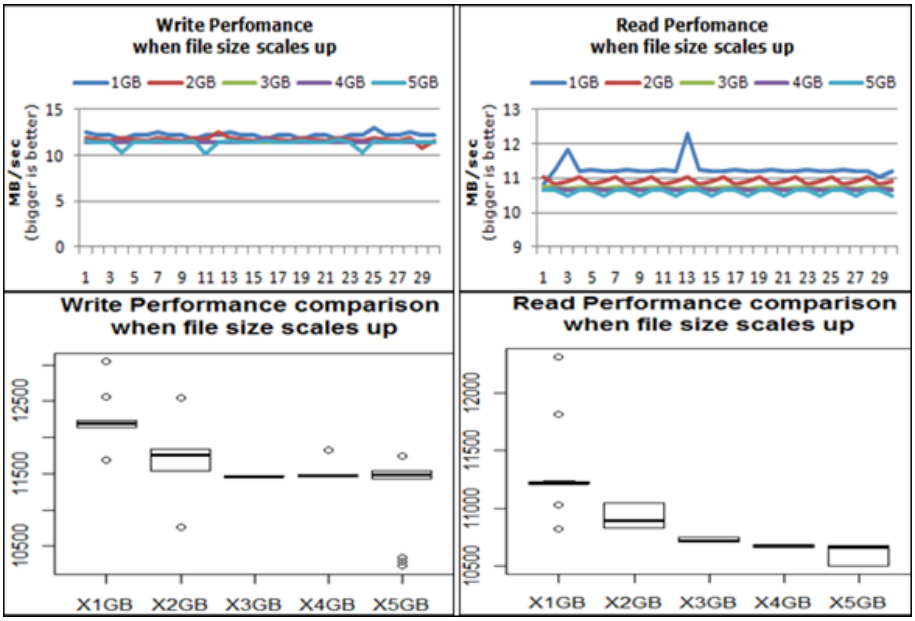


Figure 6.14: Line graphs and box plots for write and read performance comparison when file size scales up.

Sample OSD node failure

```
root@addisu-mon1:~# ceph -s
2011-05-18 07:23:32.279024    pg v9363: 792 pgs: 244 active+clean, 548
 root@addisu-mon1:~# ceph -s
2011-05-18 07:23:32.279024    pg v9363: 792 pgs: 244 active+clean, \
548 active+clean+degraded; 15389 MB data, 33998 MB used, \
2750 GB / 2790 GB avail; 2756/7746 degraded (35.580%)
2011-05-18 07:23:32.280916   mds e5: 1/1/1 up {0=up:active}, 1 up:standby
2011-05-18 07:23:32.280952   osd e7: 3 osds: 2 up, 3 in
2011-05-18 07:23:32.281007   log 2011-05-18 07:22:59.118506 \
mon0 10.0.0.8:6789/0 10 :  [INF] osd2 10.0.0.3:6800/31766 failed \
(by osd1 10.0.0.2:6800/30165) 2011-05-18 07:23:32.281098 \
 mon e1: 1 mons at {0=10.0.0.8:6789/0}
root@addisu-mon1:~#
```

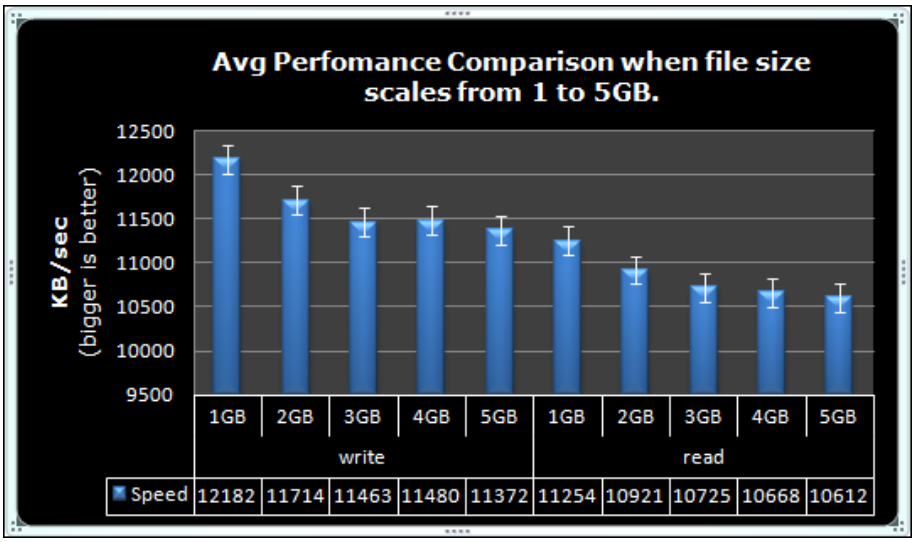


Figure 6.15: Avg write/read performance when file size scales up from 1GB to 5GB in multiple of 1.

Chapter 7

Data Analysis

The main purpose of this research is to investigate the performance, scalability, and reliability of a distributed file system, ceph. The problem stated in the 'Problem Statement' section of Introduction chapter should be answered depending on the result found from ceph benchmarking. The benchmarking and result collection are already done in the previous chapter. Boxplots and MS Excel line and bar graphs are used to show the results. The three types of graphs used to express results are all important as they all give different kind of information about the samples collected. If, for example, one need to know how the data look like at every single point and its trend, line graph is better to use. But, if one need to have general summary of the whole data (like min, max, median and the 25, 50, and 75 percent quartiles of the raw data plus the outliers if any), boxplot is the best choice. Bar graphs are also very good to compare single values of different data (like to compare average, min, max, STDEV, MEM, CONF, etc...). The meaning of the statistical terms and boxplot values are described in the result chapter.

The results found are divided according to the three main conditions set to conduct the experiments. The data analysis will also be divided according to those conditions excluding the third experiment since it doesn't, actually, need any data analysis. The third experiment is all about investigating ceph reliability by testing ceph tolerance when any node from its cluster fails. That is already proven that ceph is reliable in that respect.

7.1 Experiment type 1 result analysis.

Experiment type 1 is all about investigating ceph performance and scalability when the number of client machines using the file system scales up with common record and file size. Fortunately, the results found in this experiment do not create any complication and hence makes it easy for analysis. Clear performance degradation has been observed when the number of client nodes scales up. One could easily identify this fact by looking at the line graphs and boxplots shown in the result chapter (see Figures from 6.4 - 6.9). Even because of the high value gaps between results from 1VM and the rest 5 different number of VMs (i.e. 10, 20, . . . , 50), it was very difficult to show all 6 type of results using a single graph or boxplot. That is why some of the graphs and boxplots are constructed by excluding the result found from single client (or VM) for better visibility and readability.

It is also observed that the degradation trend follows very close to exponential function starting from 10 clients to 50 clients (See figure 7.1).

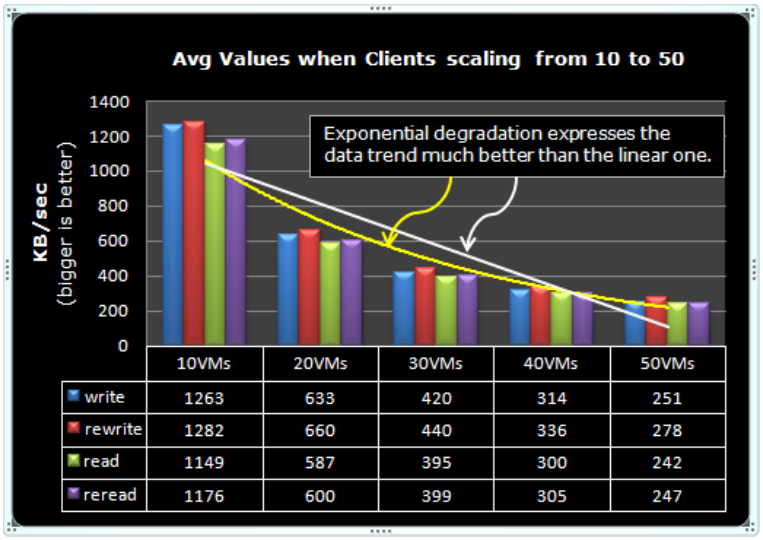


Figure 7.1: A trend of exponential performance degradation when number of clients scales up.

The margins of errors for the above result are shown in Figure 6.10 and 6.11. The standard error of mean and confidence interval values show that the actual mean of the write/read speeds in all cases are not significantly far from the sample mean values. For example; in 95 percent confidence we can say that the true mean of the write speed when 10 clients are used is between 1260.01 (1263 - 2.99) and 1265.99 (1263 + 2.99) which is good. The standard error of mean in this case

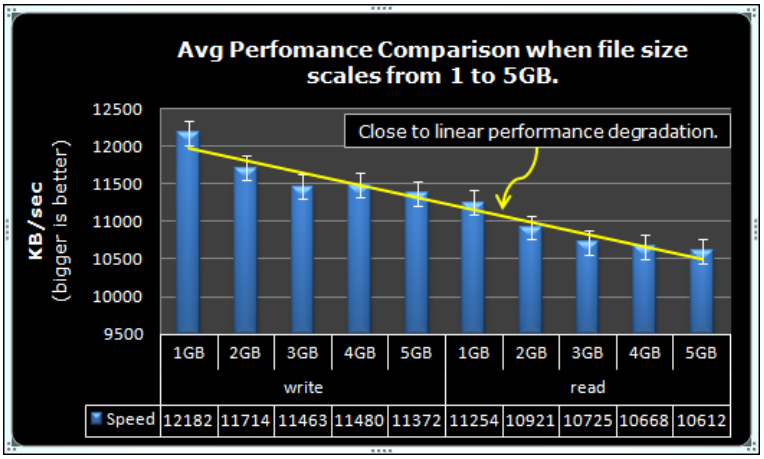


Figure 7.2: A trend of linear performance degradation when file size scales up.

is 8.36 (which is little more than two times of the confidence interval value). Similarly, in all other cases, the margins of errors are not too significant to doubt the results found in the experiments.

7.2 Experiment type 2 result analysis.

Experiment type 2 is all about investigating ceph performance and scalability when a single client is using the file system by scaling up the file size it uses to write and read. The result found in this experiment clearly shows again performance degradation. However, this time the degradation trend doesn't follow exponential function unlike shown in the case of experiment 1, rather it follows close to linear function (see figure 7.2).

Chapter 8

Discussion

NFS, SAN, and NAS are the most common file system and storage solutions nowadays in many datacenters. However, all of them are block-based systems and are all vulnerable to single point of failure and bottleneck of file I/O as they don't have load balancing mechanism. Distributed file systems solve those two big challenges faced by all NFSs for many years. DFSs are mainly designed for HPC (High Performance Computing) centers to allow parallel access to hundreds and thousands of clients and manage a lot of MDSs and OSDs with ease. The huge achievements of most of the DFSs are their ability to split metadata from actual data by implementing object-based storage systems rather than using the traditional block-based unlike NFS, SAN, and NAS are based.

More than the common achievements made by almost all DFSs, ceph has some especial and unique approach and features in its architecture. Intelligent and dynamic MDSs and OSDs, dynamic subtree management, and central monitoring nodes (MONs), are some of the very astonishing features ceph has implemented in its architecture which makes it different from other DFSs. That's why it is picked for the investigation in this research.

In this chapter we will discuss the overall project process from the beginning till the end in a summarized manner. In general we will go through the following key points:

* Challenges and limitations
* Thesis summary
* Retrospective study of approach
* Unexpected results

* Reproducibility, Repeatability, and predictability

i **Challenges and limitations**

Deciding a type of study to conduct by itself was a challenging scenario; and so making problem statement and designing approach were also the challenges faced in the very beginning of this research work. Due to three main reasons: 1) the nature of DFSs 2) lack of expertise and documentation as ceph is a new software which is still under heavy development, and 3) resource and time constraints.

Investigating performance, scalability and reliability of a distributed file system in general is a complex task. The nature of the file system by itself is not something similar to the traditional NFS, CIFS or samba. It is not something which can satisfactorily be done with a small cluster of machines and with few numbers of clients. It requires larg number of cluster of machines with very big storage size, at least hundreds of clients, and time to understand how to setup, install and configure it; all before starting the benchmarking. Due to the above three major limitations in the process of the project, tough decisions had to be made what type of study to conduct: comparative or investigative; which affects the problem statement, and approach.

In the beginning; the plan was to do comparative study. Selecting the most common and recent DFSs and compare with ceph to find out where it stands in performance and scalability. From the many DFSs out there, pNFS and Lustre had been chosen for the comparative study. pNFS Parallel Network File System is the advancement of NFS which is NFSv4.1. Similar to ceph, it is still under heavy development. And, Lustre is a stable DFS which is very popular and being used in many HPCs. That is why those two DFSs had been chosen (i.e. one is under heavy development like ceph, and the other one is stable and popular).

However; after a lot of struggle to make it happen and include both of them in the comparative study for a better result, it didn't work out. Both pNFS and Lustre had to be patched to the Linux kernel to install them. The Linux kernel configuration, compilation, and installation was one of the challenges due to lack of prior experience in the area. It took almost a week to finally patch pNFS successfully. Unfortunately, pNFS is under heavy development and currently broken and couldnt proceed with the configuration (see Appendix A).

After this point; decision was made to drop the planned comparative study. Not only because pNFS software is broken, but also the total time left for the deadline of the project wouldn't have been enough to proceed with any comparative study. Because; to conduct a comparative study, a minimum of two DFSs setup, installation, configuration, benchmarking, result analysis and comparison could have been required; which is beyond possible within the given time. Then, both pNFS and Lustre were dropped from the comparative study and concentrated on ceph investigation. Accordingly, the problem statement of the research and approach to answer the problem were redesigned.

ii **Thesis summary**

The whole thesis was restructured after the type of study to be conducted had been decided (whether comparative or investigative). As it is explained above; investigative study was the final decision; which is investigating the performance, scalability, and reliability of ceph. This sub-section goes through all chapters and discusses the choices made in every step briefly.

The Introduction chapter of this thesis provides the following important information:

* The motivation of the author to do this research and why it is important.
* File system overview and overview of ceph in particular.
* The type of problem to be tackled and why it is a problem.
* How the problem is going to be solved and approached.
* The goal and contribution of this research work.

In the background and related work chapter (chapter 2), it is discussed what file systems are all about and their groups: local, network, and distributed file systems. From each group, the most common file systems are discussed; especially, those which are open sources and latest file systems. In the DFS section, the most recent, open source, and popular DFSs are discussed by comparing their architectural differences with ceph. Benchmarking a file system and IOzone benchmarking tool are also discussed in chapter 2. Ceph and its architecture is discussed in chapter 3 in detail. The methodology chapter explains how to install and configure ceph in detail.

Chapter 5 is all about Experiment Setup. Resources used and the

lab topology are shown here. The type of benchmarking conditions, number and type of clients, IOzone options, and Perl scripts used are discussed. Even though IOzone has many options, write/rewrite and read/reread speeds are measured in all ceph benchmarking tests. One common record size is also used in all cases. By default, IOzone has 13 record sizes (4, 8, 16, ..., 16384 in KB) for each file size test from 64KB to 512MB. For simplicity and to save time the average record size (256KB) is used.

The experiment setup is divided into three main types. 1) Investigate ceph performance and scalability when number of clients scales up. 2) Investigate ceph when file size scales up. 3) Ceph reliability test by failing one of the OSD nodes.

In type 1 benchmarking condition, 50 KVM virtual machines are created to use them as ceph clients. This was the maximum possible number of clients that could be created on the provided Dell PowerEdge R610 server due to disk limitation. For better result and reflect real situations, it was attempted to rent many VMs from Amazon CC. It was, of course, possible to rent; but was not possible to mount ceph due to the fact that Amazon firewall blocks ceph port number; which is 6789 by default. Hence; in condition type 1; ceph benchmarking measurements were taken by scaling the number of clients (VMs) from 1 to 10, then to 20, ..., 50. Due to time constraints, in all tests in this condition a single and common file size of 1GB for the IOzone option was used. A common file size of 1GB was decided after some test trials and time calculations. File size of 5GB, 10GB, 20GB from 20 VMs were tested and found that they all took very long time to finish the test. For example, 20GB of file size from 20 VMs took more than 48 hours to finish the testing. Therefore, it wouldn't have been possible to get 30 iterations test results from each condition type within the given time. When 1GB was decided to use for IOzone file size option, the IOzone rule 1 was taken into consideration which says it should be bigger than the RAM size to avoid client side buffer cache effect.

In type 1 benchmarking condition, only 1 VM was used by scaling up file size from 1 to 5GB in multiple of 1. Again here; small file sizes were used to save time. However; except ceph couldn't have been stressed by huge file sizes because of the small file sizes used in the benchmarking, 1GB file size is enough to get reliable results anyways.

In this same chapter, three Perl scripts are discussed. The first one

is used to prepare the 50 VMs for the benchmarking (like: create 50 folders in ceph file system root directory for the 50 VMs, install ceph and iozone3 and mount ceph on all VMs). The second Perl script is used to do the benchmarking according to the number of clients to be used at a specific time. It saves cron jobs in all client VMs to start the IOzone benchmarking run at the same time. It receives flags and arguments at the command line to determine the number of clients and /etc/crontab timing. And; the third Perl script is used to collect the outputs from all VMs and organize it in a column. The complete codes of the three Perl scripts are found in Appendix D.

In chapter 6 we get the results of the three experiment types. Excel line and bar graphs and R boxplots are used to show the results. All three types of graphs used are important when we see the type of information we get from each of them. They are all explained in the same chapter. Except there were some outliers, the overall result clearly talks about the performance and scalability of ceph very well. And, the type 1 and 2 experiment results contradict the claims ceph argues. It showed very poor performance degradation when number of clients scales up and when file size scale up either in the experiment type2. However, in the reliability test (experiment type 3), it proved that ceph is highly reliability. The descriptive statistics in the Analysis chapter showed exponential degradation in the experiment type 1 result; and linear degradation in the experiment type 2. The monitoring tools used to monitor the ceph client and server side when the benchmarking is ongoing also discussed in this chapter.

iii **Retrospective study of approach**

Ceph benchmarking went more or less well with 1GB of file size and from a maximum of 50 clients. However; due to time, resource, and expertise limitations, number of clients and file sizes used in the benchmarking were reduced very significantly. Because of that ceph file system was not stressed enough by heavy file sizes from at least hundreds of clients in the benchmarking experiments. The results found, actually, are highly consistent as their margins of errors are very less. However, it would have been much better if ceph had been tested with bigger file sizes and from more number of clients. At least up to 75 percent of its total storage size is used.

In addition to that, without comparing it with other similar products, it is also hard to evaluate its performance and scalability

and answer the question where it stands compare to other DFSs. Hence, comparing ceph with other DFS products (like Lustre and pNFS) is very important and highly recommended to exactly know how well ceph performs. This was, of course, the first plan of this project. This is also discussed in chapter 10 (Future Work).

To reflect the real situation of use of DFSs, benchmarking ceph from nodes (clients) outside of ceph cluster LAN would have been much better.

iv **Unexpected results**

If it is according to the great architectural approaches of ceph, one normally expects strong performance and very high scalability. However, in the contrary, getting exponential degradation in performance when number of clients scale up is something beyond one's imagination.

Ceph is, of course, special and unique in some of its features and already showed better abilities in some areas. For example, it is possible to expand its cluster and storage size as much as one needs; which many DFSs are not able. Its reliability is already tested in the experiment type 3, and showed high reliability. But, as impressive as it is, it has also some drawbacks as per the research findings of this paper in addition to the exponential performance degradation observed when scales number of clients up.

Ceph storage usage is one of its drawbacks. Surprisingly, total storage size degrades more than double of data size in many cases. It charges you big storage size to maintain its reliability. It actually uses btrfs file system to format the provided disks. Btrfs is copy-on-write file system. Similar to ceph, Btrfs is also under heavy development. However, it is even taking more than double of the actual size. It is like, on average to save and maintain N Kilobyte of data, ceph will charge you around (3.5 * N) of storage size which is very expensive. See the sample output of 'ceph -w' in section 6.1.2.

When looking at the great architectural benefits of ceph, it is hard to expect poor scalability in regard to performance when number of clients and file sizes scale up.

v **Reproducibility, Repeatability, and predictability**

Even though ceph is still under heavy development, the setup, installation and configuration in this research didn't create very significant challenges except small problems. Its simplicity to expand

its size and monitor its activity is quite impressive. It supports heterogeneous hardware and recent Linux kernel version (above 2.6.34) includes ceph. The results also show clear trend and meaningful graphs that the whole project could be reproduced.

Since the trends of the results found are consistent and show clear line of graph except very few outliers (which are excluded from the average calculations), its predictability is high. If we take the first experiment type, for example, the graph of the average write/read graph indicates there is an exponential degradation. If there had been 60VMs, 70VMs ...etc, we can easily predict the result by following the exponential graph.

The project's repeatability is also high. If many more repeated tests had been taken under same condition, similar result would have been found as margins of standard errors and confidence interval values are very less. Hence, the results were repeatable.

Chapter 9

Conclusion

Ceph has achieved a milestone in the development of distributed file systems by adding some special and unique concepts and features. Its maximum separation of metadata from actual data, its dynamic subtree management system, its intelligent and dynamic OSDs and MDSs are some of the great achievements made by ceph.

Since it doesn't put limitation on the number of metadata servers (MDSs) and/or object based storage nodes (OSDs), it can be expanded extremely which pNFS even lacks it. If the extremely scalability which ceph claims is in regard to extremely expandable, it is true. However, as per the findings of this paper, its performance when number of clients scales up degrades very significantly. Performance degradation when number of clients scale up is something expected in almost all file systems. However, exponential degradation is something beyond imagination.

Ceph assumes node failure as a norm not as an exception. This is a great architectural approach to be highly reliable. Ceph reliability is proven in this research, but with high storage price. It uses more than triple times of the actual data available.

Ceph, actually, is still under heavy development. A lot of bug fix is ongoing very rapidly and its version is also changing every 20 days on average. That could be is one of the reasons behind its high degradation when number of clients scales up. Hence, putting in mind its architectural advantages, future work is recommended to reinvestigate it and compare its performance, scalability, and reliability with other DFSs; especially after the software becomes stable and ready for production use.

Chapter 10

Future Work

Since there is a high demand of high performance, scalable, and reliable file system which can handle a lot of storage sizes and thousands of clients, researching and working on distributed file system is highly essential. The current and future file system is highly dependent on distributed file systems. NFS, SAN, and NAS are becoming obsolete because of their limitation in performance and scalability. It is because they impose the whole workload on a single server; and data and metadata are also seated together since they all use traditional block based storage system.

Distributed file systems solves some of the obstacles faced by NFS and other similar file systems; especially, by decoupling metadata from actual data and by using the newly immerged object based storage system. Among the families of DFSs, ceph is one of the latest products with having some more unique and especial architectural approaches.

It is hard to conclude about ceph performance and scalability depending only on this project result which is affected by the limitations occurred. The fact that ceph is currently being under heavy development also has some effect on its performance.

Hence; continuing the investigation of ceph performance and scalability by considering the following three key points are highly recommended:

* Ceph cluster setup and clients are recommended to be in different subnets to reflect real life situation.
* Use as many clients as possible; at least not less than hundred. The VMs are recommended to be distributed in different servers located in different subnets to be as representative of real life situ-

ation as possible.

* Allocate enough time and resource to be able to benchmark ceph using as bigger file size as possible and from as many clients as possible.

Appendix A

More about pNFS and Lustre

pNFS (NFSv4.1) and Lustre:

In the beginning of this project, it was planned to consider comparing ceph with one of the very popular and highly used distributed file systems which is Lustre and with one of the very latest product and at the same time under heavy development similar to ceph which is pNFS. However, while on the process after going through the installation and configuration of both Lustre and pNFS, I came to know that the time frame given for the project is too short to accomplish three distributed file systems' (ceph, pNFS, and Lustre) installation, configuration, benchmarking, analysis, and comparison. That is why both pNFS and Lustre was dropped and left for future work. In this section, we will discuss the pNFS installation only.

In the first place, both pNFS and Lustre installation need Linux kernel configuration, compilation and installation after patching them. This, by itself, was a challenge for me since I didn't have kernel configuration experience earlier. Since I was very keen to include both of them in the project to compare with ceph, I decided to allocate some time to do it. I started with pNFS; and after some trial and error I was success to boot-up a machine with pNFS enabled kernel. Since I didn't get a complete documentation to do the pNFS kernel configuration, it took me days not only hours. That is why I want to share my experience. See below the steps I used to do the kernel patching (Debian and Ubuntu way): s

```
$ apt-get install kernel-package
$ apt-get install libncurses5-dev
```

```
$ apt-get install fakeroot
$ apt-get install build-essential
```

Or, install all of them in one go,

```
$ apt-get install kernel-package libncurses5-dev fakeroot build-essential
$ cd /usr/src/
```

* Get pnfs-enabled Linux kernel from git repository (which needs to be configured, compiled and installed).

  ```
  $ git clone git://git.linux-nfs.org/projects/bhalevy/linux-pnfs.git
  ```

* To put the output files including .config file in a different location when using 'make' command create a separate folder in your home directory.

  ```
  $ mkdir -p /home/name/build/kernel/
  ```

* Make a soft link to your kernel source directory for safety.

  ```
  $ ln -s linux-pnfs/ linux
  ```

* Then, use 'linux' directory to configure, compile, and install the new kernel. Now, delete or clean all object files if any; by executing 'make clean' command.

  ```
  $ cd linux
  $ make O=/home/addisu/build/kernel clean
  ```

* To delete everything including previous .config file except your sources which makes it properly ready for the new configuration and compilation, execute 'make mrproper' command.

  ```
  $ make O=/home/addisu/build/kernel mrproper
  ```

* The next very important step is to copy your current .config file to the new .config file; it copies your current kernel configuration to the new .config file, and then you can continue from your current .config file to configure (or patch) the pNFS modules. Otherwise, the Linux kernel system will consider the default for all un-configured options.

```
$ cp /boot/config-2.6.35-28-server .config
```

* You can configure kernel in several ways using one of the following options: "make config", "make menuconig", "make oldconfig", etc...

  I first used 'make config' command to configure the kernel and took me very long time to configure it and I even missed patching the pNFS modules two time while on the process and hence failed. I, finally, came to know the importance of menuconfig. There is huge difference. Using 'make menuconfig' command you can have all options in a menu in GUI format and can go back and forth to enable whichever module you want.

```
$ cp /boot/config-2.6.35-28-server .config
```

* Make sure that the following latest pNFS branches are included in your .config file while configuring.

```
CONFIG_NETWORK_FILESYSTEMS=y
CONFIG_NFS_FS=m
CONFIG_NFS_V4=y
CONFIG_NFS_V4_1=y
CONFIG_PNFS=y
CONFIG_NFSD=m
CONFIG_PNFSD=y
# CONFIG_PNFSD_LOCAL_EXPORT is not set
CONFIG_SPNFS=y
```

See Figure A.1 for the pNFS (NFSv4.1) options in the 'Network File Systems' menu lists of the Linux-pNFS kernel configuration.

After configuration, the next step is compilation. Before doing that you need to clean the area from any other conflicting .config file or object files. Since compilation takes very long time, before doing the steps below, you must be sure that you have done patching the necessary pNFS(NFSv4.1) options according the above lists.

* You need to go back to the linux-pnfs folder and do cleaning and deleting everything.

```
$ cd ../linux-pnfs/
$ make clean
$ make mrproper
$ cd ../linux
```

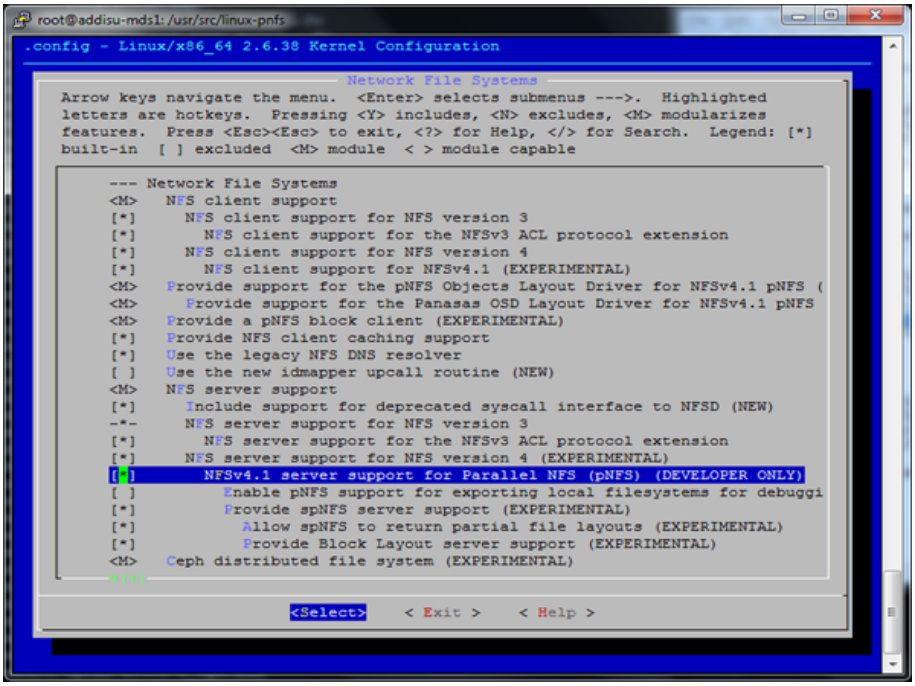


Figure A.1: pNFS(NFSv4.1) options in the Linux-pNFS kernel.

```
$ make O=/home/addisu/build/kernel
```

* This will compile the kernel and takes very long time. You better go and have tea or take a nap; it takes more than two hours.

```
$ make O=/home/addisu/build/kernel modules_install install
```

* This will install the new kernel.

```
$ cp /home/addisu/build/kernel/arch/x86_64/boot/bzImage\
 /boot/vmlinuz-2.6.38-pnfs
```

* Copy the kernel image to the boot directory.

```
$ update-grub
$ reboot
```

Enjoy....

You can check the kernel version by executing 'uname -r'. You must see something like this:

Linux-2.6.38-pnfs.

Now you have latest kernel version with pNFS enabled. The next step is configuration and it is relatively easy. The URL below [6] shows the configuration step by step which is almost straight forward.

> http://git.linux-nfs.org/?p=bhalevy/linux-pnfs.git;
> a=commitdiff_plain; h=pnfs-all-2.6.35
> -2010-09-14;hp=v2.6.35

The bad news is: pNFS is not stable; it is under heavy development and unfortunately pNFS daemon is currently broken. This is what stopped me from proceeding after finishing installation and configuration.

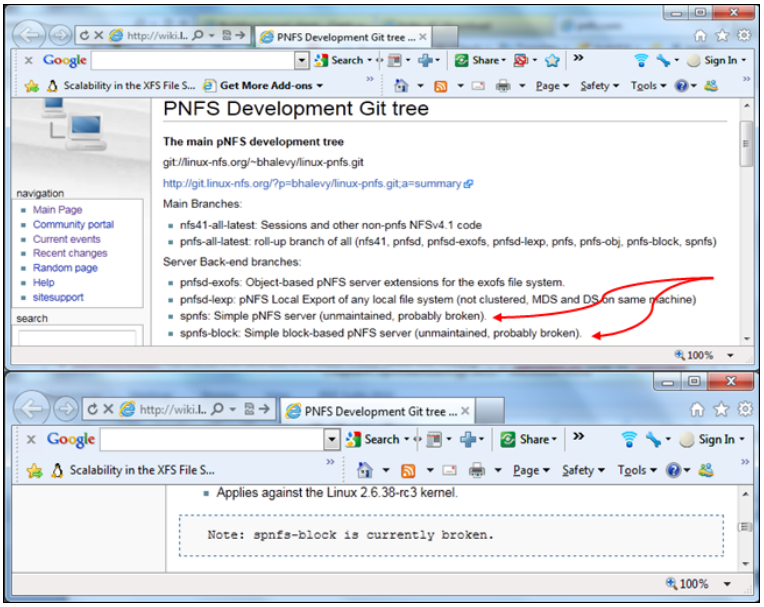


Figure A.2: pNFS web site showing the currently broken system.

The other challenge I faced was the effect of different hardware architectures in the kernel installation. Since the machines used in this project have three different types of hardware architectures, the above steps didn't work for all of them. Actually, it worked only for one of them. I installed pNFS on three similar machines; but unable to install it on the other two models. It was not clear for me why it didn't work for the other two models even though all of them support x86_64 architecture.

Appendix B

IOzone sample executions and outputs

IOzone sample executions outputs:

1. Sample output of IOzone auto mode run with read/re-read, write/re-write, and excel options.

Sample IOzone output for 'iozone -Ra -i 0 -i 1' command

```
1  Iozone: Performance Test of File I/O
2              Version $Revision: 3.308 $
3              Compiled for 64 bit mode.
4              Build: linux
5
6          Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
7                  Al Slater, Scott Rhine, Mike Wisner, Ken Goss
8                  Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
9                  Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
10                 Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy,
11                 Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root.
12
13         Run began: Fri Apr 29 08:23:52 2011
14
15         Excel chart generation enabled
16         Auto Mode
17         Command line used: iozone -Ra -i 0 -i 1
18         Output is in Kbytes/sec
19         Time Resolution = 0.000001 seconds.
20         Processor cache size set to 1024 Kbytes.
21         Processor cache line size set to 32 bytes.
22         File stride size set to 17 * record size.
23                                                    random  random    bkwd
24             KB  reclen   write rewrite    read    reread    read   write    read
25             64       4  874936 1304314  1879725  2467108
26             64       8 1232453 1722886  3057153  3203069
27             64      16 1304314 1828508  3363612  4018152
28             64      32 1392258 1947927  3541098  3588436
29             64      64 1357066 2006158  3541098  4274062
30            128       4 1032829 1373754  2326073  2511022
31            128       8 1319723 1802755  3106789  3359523
32            128      16 1391557 1885042  3468030  3657016
```

```
            128     32 1471662 2035099  3560017  3759450
            128     64 1539168 2098744  3759450  4267461
            128    128 1539168 2211113  4012317  4267461
            256      4 1158540 1506359  2692393  2880164
            256      8 1201314 1752173  3087188  3275543
            256     16 1384034 1829808  3285566  3454704
            256     32 1455317 1829808  3545976  3667079
            256     64 1523457 2205688  4350555  4755158
            256    128 1588832 2192179  3935921  4197489
            256    256 2242541 2371308  4652146  5022044
            512      4 1379416 1551867  2548017  2668325
.
.
         262144   16384  853725 1597317  1923905  1933272
          524288      64  955568 1802587    11070  4179057
          524288     128  729899 2148862  5223658  5393797
          524288     256  454707 2143958  5321640  5453388
          524288     512  835694 2165951  5362623  5496653
          524288    1024  790184 1914416  4754975  4884365
          524288    2048  786253 1591485  2206933  2219574
          524288    4096  595766 1560664  1919478  1926247
          524288    8192  841328 1568709  1912071  1917092
          524288   16384  953764 1576990  1923471  1929872

iozone test complete.
Excel output is below:

"Writer report"
      "4"  "8"  "16"  "32"  "64"  "128"  "256"  "512"  "1024"  "2048"  "4096"  "8192"
"16384"
"64"   874936  1232453  1304314  1392258  1357066
"128"   1032829  1319723  1391557  1471662  1539168  1539168
"256"   1158540  1201314  1384034  1455317  1523457  1588832  2242541
"512"   1379416  1652157  1759070  1533034  1616100  2007358  1809465

1875849

"1024"   1499741  1828574  1428415  2165014  1635675  2316837  2370545
1615374
2226749

"2048"   1161774  1298592  1400881  2218586  2288326  1605053
2263006 2338157  1496033  1597590

"4096"   1005623  1256407  1356290  2090820  1690465  2196407
2191364  2144586  1430248  1247285  1074762

"8192"   134763  109378  99797  89222  61063  30081  98591
125043  200381  142281  159774  91419

"16384"   156805  129458  149634  141802  158710  148947
148742  121675  150671  147644  122368  184644  257060

"32768"   0  0  0  0  159549  171707  223387  203256  208404
172678  179733  252888  506702

"65536"   0  0  0  0  242586  92312  306324  290138  133096
309438  267496  390155  501922

"131072"   0  0  0  0  186777  504833  527692  173006  211844
421407  262693  349789  609770
```

```
95  "262144"   0  0  0  0  355594  689299  650986  345944  690994
96  592094  652224  751995  853725
97
98  "524288"   0  0  0  0  955568  729899  454707  835694  790184
99  786253  595766  841328  953764
100 "Re-writer report"
101       ...
```

2. Figure B.1 shows Sample output for the auto mode 'IOzone -a' command and the meaning of the first two columns.

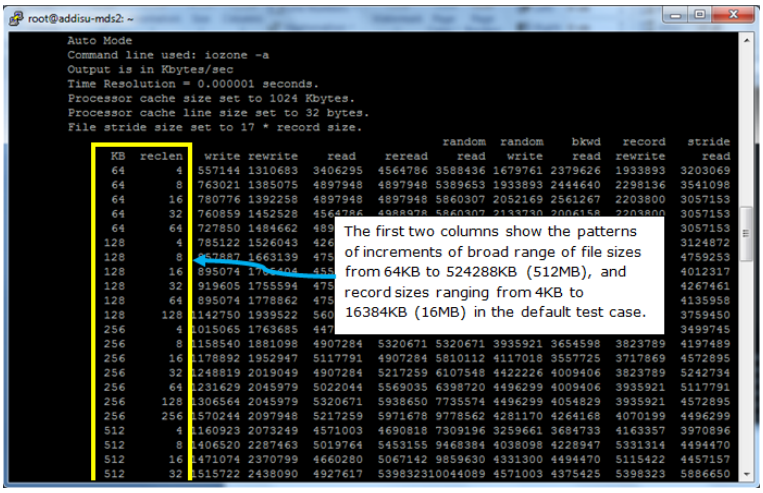


Figure B.1: Sample output of Auto mode default **'iozone a'** command.

3. Figure B.2 shows the IOzone excel compatible output.

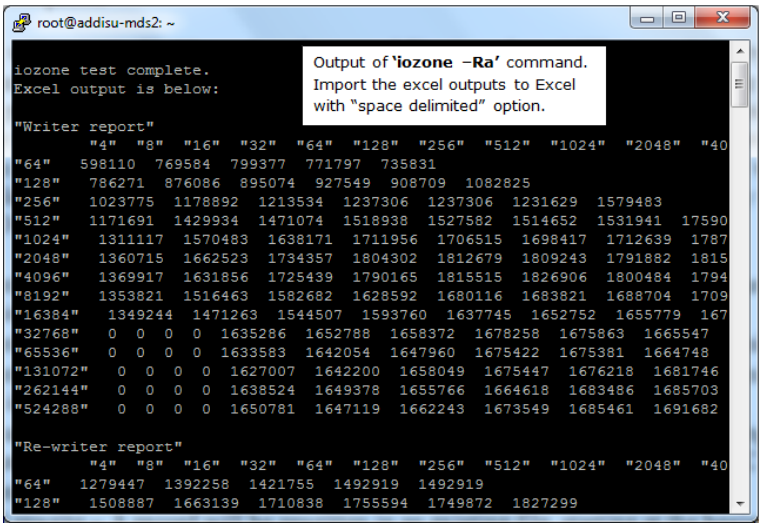


Figure B.2: Sample output of IOzone command with excel generation option.

4. Sample IOzone output with single record size option.

IOzone output for 'iozone -i 0 -i 1 -r 256 -s 80G' command

```
root@ceph-client:/mnt/ceph# cat 1vm_500_80_test.txt
        Iozone: Performance Test of File I/O
                Version $Revision: 3.308 $
                Compiled for 64 bit mode.
```

```
        Build: linux

    Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
            Al Slater, Scott Rhine, Mike Wisner, Ken Goss
            Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
            Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
            Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy,
            Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root.

    Run began: Sat Apr 30 17:18:32 2011

    Record Size 256 KB
    File size set to 83886080 KB
    Command line used: iozone -i 0 -i 1 -r 256 -s 80G
    Output is in Kbytes/sec
    Time Resolution = 0.000001 seconds.
    Processor cache size set to 1024 Kbytes.
    Processor cache line size set to 32 bytes.
    File stride size set to 17 * record size.
                                                random  random    bkwd
            KB  reclen   write rewrite    read    reread    read   write    read
    83886080     256   11436    1830    10610    10675

iozone test complete.
```

Appendix C

Commands and sample executions

Commands and sample executions:

1. Creating and mounting btrfs file system sample execution.

First install the btrfs tools and build-dep;

```
$ apt-get install btrfs-tools build-dep
```

sample mkfs.btrfs run and btrfs mount

```

root@addisu-osd1:~# mkfs.btrfs /dev/sdb7

WARNING! - Btrfs Btrfs v0.19 IS EXPERIMENTAL
WARNING! - see http://btrfs.wiki.kernel.org before using

fs created label (null) on /dev/sdb7
        nodesize 4096 leafsize 4096 sectorsize 4096 size 250.00GB
Btrfs Btrfs v0.19
root@addisu-osd1:~# mount /dev/sdb5 /mnt/btrfs_osd1_sdb5
root@addisu-osd1:~# mount /dev/sdb6 /mnt/btrfs_osd1_sdb6
root@addisu-osd1:~# mount /dev/sdb7 /mnt/btrfs_osd1_sdb7
root@addisu-osd1:~# df -hT
Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/sda1     ext4     71G  985M   66G   2% /
none      devtmpfs    997M  212K  997M   1% /dev
none         tmpfs   1003M     0 1003M   0% /dev/shm
none         tmpfs   1003M   36K 1003M   1% /var/run
none         tmpfs   1003M     0 1003M   0% /var/lock
/dev/sdb5    btrfs    250G   56K  250G   1% /mnt/btrfs_osd1_sdb5
/dev/sdb6    btrfs    250G   56K  250G   1% /mnt/btrfs_osd1_sdb6
/dev/sdb7    btrfs    250G   56K  250G   1% /mnt/btrfs_osd1_sdb7
root@addisu-osd1:~# exit

```

2. ssh password-less access sample execution:

```
$ ssh-keygen  (being on the master node)

root@addisu-mon1:~# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
d6:05:5c:02:8c:c9:d4:d2:de:ba:91:c5:87:96:4c:de root@addisu-mon1
The key's randomart image is:
+--[ RSA 2048]----+
|    o.*.oo..     |
|     = + oo      |
|      o * +.     |
|       ..X.E     |
|       S=..      |
|      .+         |
|       o         |
|       .         |
|                 |
+-----------------+

Then,
$ ssh-copy-id root@slave_node (do the same to all slave nodes)

root@addisu-mon1:~# ssh-copy-id root@mds1
The authenticity of host 'mds1 (10.0.0.6)' can't be established.
RSA key fingerprint is f2:4f:af:61:22:ba:94:0d:a2:71:10:b2:67:b9:e9:95.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'mds1,10.0.0.6' (RSA) to the list of known hosts.
root@mds1's password:
Now try logging into the machine, with "ssh 'root@mds1'", and check in:

  .ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.

root@addisu-mon1:~# ssh-copy-id root@osd1
The authenticity of host 'osd1 (10.0.0.1)' can't be established.
RSA key fingerprint is 4e:cf:d8:dd:9e:be:47:11:74:e8:7d:62:e9:ff:ba:08.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'osd1,10.0.0.1' (RSA) to the list of known hosts.
root@osd1's password:
Now try logging into the machine, with "ssh 'root@osd1'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.

root@addisu-mon1:~#

Confirm that the password-less root access is working:

root@addisu-mon1:~# ssh root@mds1
Linux addisu-mds1 2.6.35-22-server #33-Ubuntu SMP Sun Sep
19 20:48:58 UTC 2010 x86_64 GNU/Linux Ubuntu 10.10

Welcome to the Ubuntu Server!
```

```
 * Documentation:  http://www.ubuntu.com/server/doc
Last login: Mon Apr 18 19:12:17 2011 from 10.0.0.8
root@addisu-mds1:~#

```

3. Sample run of mkcephfs command to create ceph file system:

sample mkcephfs use

```
root@addisu-mon1:~# /sbin/mkcephfs -c /etc/ceph/ceph.conf -a --mkbtrfs
temp dir is /tmp/mkcephfs.Sn0LCB2mSH
preparing monmap in /tmp/mkcephfs.Sn0LCB2mSH/monmap
/usr/bin/monmaptool --create --clobber --add 0 10.0.0.8:6789
--print /tmp/mkcephfs.Sn0LCB2mSH/monmap
/usr/bin/monmaptool: monmap file /tmp/mkcephfs.Sn0LCB2mSH/monmap
/usr/bin/monmaptool: generated fsid
8f1672f7-6534-f970-3dc3-6fe70999a516 epoch 1
fsid 8f1672f7-6534-f970-3dc3-6fe70999a516
last_changed 2011-04-20 11:35:31.675649
created 2011-04-20 11:35:31.675649
0: 10.0.0.8:6789/0 mon.0
/usr/bin/monmaptool: writing epoch 1 to /tmp/mkcephfs.Sn0LCB2mSH/monmap

(1 monitors)
=== osd.0 ===
pushing conf and monmap to osd1
umount: /data/osd0: not mounted
umount: /dev/sdb5: not mounted

WARNING! - Btrfs Btrfs v0.19 IS EXPERIMENTAL
WARNING! - see http://btrfs.wiki.kernel.org before using

fs created label (null) on /dev/sdb5
        nodesize 4096 leafsize 4096 sectorsize 4096 size 250.00GB
Btrfs Btrfs v0.19
Scanning for Btrfs filesystems
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
 ** WARNING: 'filestore btrfs snap' is enabled (for safe transactions,
            rollback), but btrfs does not support the SNAP_CREATE_V2 ioctl
            (added in Linux 2.6.37).  Expect slow btrfs sync/commit
            performance.
2011-04-20 11:35:23.981540 7f2852ab6720 created object
store /data/osd0 journal /data/osd0/journal for osd0
fsid 8f1672f7-6534-f970-3dc3-6fe70999a516
creating private key for osd.0 keyring /tmp/mkcephfs.Sn0LCB2mSH/keyring.osd.0
creating /tmp/mkcephfs.Sn0LCB2mSH/keyring.osd.0
collecting osd.0 key
=== osd.1 ===
pushing conf and monmap to osd2
umount: /data/osd1: not mounted
umount: /dev/sdb5: not mounted

WARNING! - Btrfs Btrfs v0.19 IS EXPERIMENTAL
WARNING! - see http://btrfs.wiki.kernel.org before using

fs created label (null) on /dev/sdb5
        nodesize 4096 leafsize 4096 sectorsize 4096 size 250.00GB
Btrfs Btrfs v0.19
Scanning for Btrfs filesystems
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
```

```
 ** WARNING: 'filestore btrfs snap' is enabled (for safe transactions,
           rollback), but btrfs does not support the SNAP_CREATE_V2 ioctl
           (added in Linux 2.6.37).  Expect slow btrfs sync/commit
           performance.
2011-04-20 11:35:37.766249 7f4f4e604720 created
object store /data/osd1 journal /data/osd1/journal
for osd1 fsid 8f1672f7-6534-f970-3dc3-6fe70999a516
creating private key for osd.1 keyring /tmp/mkcephfs.Sn0LCB2mSH/keyring.osd.1
creating /tmp/mkcephfs.Sn0LCB2mSH/keyring.osd.1
collecting osd.1 key
=== mds.mds1 ===

pushing conf and monmap to mds1
creating private key for mds.mds1 keyring
/tmp/mkcephfs.Sn0LCB2mSH/keyring.mds.mds1
creating /tmp/mkcephfs.Sn0LCB2mSH/keyring.mds.mds1
collecting mds.mds1 key
Building osdmap
 highest numbered osd in /tmp/mkcephfs.Sn0LCB2mSH/conf is osd.1
 num osd = 2
/usr/bin/osdmaptool: osdmap file '/tmp/mkcephfs.Sn0LCB2mSH/osdmap'
/usr/bin/osdmaptool: writing epoch 1 to /tmp/mkcephfs.Sn0LCB2mSH/osdmap
Generating admin key at /tmp/mkcephfs.Sn0LCB2mSH/keyring.admin
creating /tmp/mkcephfs.Sn0LCB2mSH/keyring.admin
Building initial monitor keyring
added entity mds.mds1 auth auth
auid = 18446744073709551615
key=AQCNqK5N6PdUBxAA9mGR+oT4nYoO+3aL3F3qog==
with 0 caps)
added entity osd.0 auth auth
(auid =18446744073709551615
key=AQDbqK5NYFFtOxAAzvutdpy2CD0jfIm5rRKWMg==
with 0 caps) added entity osd.1 auth
auth(auid = 18446744073709551615 key=AQDpqK5NgGuaLhAAwgS37ptqWOe++
TW5Yk2l9A==
with 0 caps)
=== mon.0 ===

pushing everything to mon1
 ** WARNING: Ceph is still under heavy development, and is only suitable for **
 **          testing and review.  Do not trust it with important data.       **
/usr/bin/cmon: created monfs at /data/mon0 for mon.0
placing client.admin keyring in /etc/ceph/keyring
root@addisu-mon1:~# exit

```

4. Sample ceph Interactive mode execution:

Sample Interactive mode use

```
root@addisu-mon1:~# ceph
ceph> osd stat
2011-04-21 01:44:23.713333 mon <- [osd,stat]
2011-04-21 01:44:23.713656 mon0 -> 'e5: 2 osds: 2 up, 2 in' (0)
ceph> auth list
2011-04-21 01:45:14.533318 mon <- [auth,list]
2011-04-21 01:45:14.533570 mon0 -> 'installed auth entries:
mon.
        key: AQDnqK5NgEN/AxAAecmjew6LZh7EJHvOyj4Png==
mds.mds1
        key: AQCNqK5N6PdUBxAA9mGR+oT4nYoO+3aL3F3qog==
        caps: [mds] allow
```

```
        caps: [mon] allow rwx
        caps: [osd] allow *
osd.0
        key: AQDbqK5NYFFtOxAAzvutdpy2CD0jfIm5rRKWMg==
        caps: [mon] allow rwx
        caps: [osd] allow *
osd.1
        key: AQDpqK5NgGuaLhAAwgS37ptqWOe++TW5Yk2l9A==
        caps: [mon] allow rwx
        caps: [osd] allow *
client.admin

    key: AQDnqK5NgGuoAhAA2Zwk6lbzU5ZR4HIBMU49/Q==
        caps: [mds] allow
        caps: [mon] allow *
        caps: [osd] allow *
' (0)
ceph> quit
root@addisu-mon1:~#
```

Sample ceph watch mode execution:

Sample ceph watch mode use

```
root@addisu-mon1:~# ceph \- w
2011-04-21 01:49:02.275322    pg v342: 528 pgs: 528 active+clean; 69 KB data,
61700 KB used,  499 GB / 500 GB avail
2011-04-21 01:49:02.276017   mds e4: 1/1/1 up {0=up:active}
2011-04-21 01:49:02.276040   osd e5: 2 osds: 2 up, 2 in
2011-04-21 01:49:02.276077   log 2011-04-21 00:21:15.473404
mon0 10.0.0.8:6789/0 1 : [INF]
mon.0@0 won leader election with quorum 0
2011-04-21 01:49:02.276136   mon e1: 1 mons at {0=10.0.0.8:6789/0}
2011-04-21 01:52:04.476858    pg v343: 528 pgs: 528 active+clean;
71 KB data, 61700
KB used, 499 GB / 500 GB avail
2011-04-21 01:52:06.877739    pg v344: 528 pgs: 528 active+clean;
71 KB data, 61716
KB used,  499 GB / 500 GB avail
2011-04-21 01:52:14.477389    pg v345: 528 pgs: 528 active+clean; 71
KB data, 61876 KB used,  499 GB / 500 GB avail
2011-04-21 01:52:39.478223    pg v346: 528 pgs: 528 active+clean; 72
KB data, 61776 KB used,  499 GB / 500 GB avail
2011-04-21 01:52:41.880569    pg v347: 528 pgs: 528 active+clean; 75
KB data, 61892 KB used,  499 GB / 500 GB avail
2011-04-21 01:53:09.481073    pg v348: 528 pgs: 528 active+clean; 77
KB data, 61868 KB used,  499 GB / 500 GB avail
2011-04-21 01:53:11.884801    pg v349: 528 pgs: 528 active+clean; 74
KB data, 61892 KB used, 499 GB / 500 GB avail
2011-04-21 01:53:14.479695    pg v350: 528 pgs: 528 active+clean; 75
KB data, 62084 KB used, 499 GB / 500 GB avail
2011-04-21 01:53:24.485305    pg v351: 528 pgs: 528 active+clean; 75
KB data, 61924 KB used, 499 GB / 500 GB avail
2011-04-21 01:53:29.483307    pg v352: 528 pgs: 528 active+clean; 77
KB data, 61996 KB used, 499 GB / 500 GB avail
2011-04-21 01:53:31.886723    pg v353: 528 pgs: 528 active+clean; 76
KB data, 62044 KB used,  499 GB / 500 GB avail
2011-04-21 01:53:34.483773    pg v354: 528 pgs: 528 active+clean; 78
KB data, 62220 KB used,  499 GB / 500 GB avail
```

```
root@addisu-mon1:~#
```

6. Sample ceph command line mode executions:

Sample ceph command line mode use

```
root@addisu-mon1:~# ceph -s
2011-04-21 01:47:34.435331    pg v342: 528 pgs: 528 active+clean; 69
KB data, 61700 KB used, 499 GB / 500 GB avail
2011-04-21 01:47:34.436024   mds e4: 1/1/1 up {0=up:active}
2011-04-21 01:47:34.436048   osd e5: 2 osds: 2 up, 2 in
2011-04-21 01:47:34.436086   log 2011-04-21 00:21:15.473404
mon0 10.0.0.8:6789/0 1: [INF] mon.0@0 won leader election with quorum 0
2011-04-21 01:47:34.436147   mon e1: 1 mons at {0=10.0.0.8:6789/0}
root@addisu-mon1:~# ceph auth list
2011-04-21 01:47:55.644820 mon <- [auth,list]
2011-04-21 01:47:55.645023 mon0 -> 'installed auth entries:
mon.
        key: AQDnqK5NgEN/AxAAecmjew6LZh7EJHvOyj4Png==
mds.mds1
        key: AQCNqK5N6PdUBxAA9mGR+oT4nYoO+3aL3F3qog==
        caps: [mds] allow
        caps: [mon] allow rwx
        caps: [osd] allow *
osd.0
        key: AQDbqK5NYFFtOxAAzvutdpy2CD0jfIm5rRKWMg==
        caps: [mon] allow rwx
        caps: [osd] allow *
osd.1
        key: AQDpqK5NgGuaLhAAwgS37ptqWOe++TW5Yk2l9A==
        caps: [mon] allow rwx
        caps: [osd] allow *
client.admin
        key: AQDnqK5NgGuoAhAA2Zwk6lbzU5ZR4HIBMU49/Q==
        caps: [mds] allow
        caps: [mon] allow *
        caps: [osd] allow *
' (0)


root@addisu-mon1:~# ceph osd stat
2011-04-21 01:48:17.764811 mon <- [osd,stat]
2011-04-21 01:48:17.764985 mon0 -> 'e5: 2 osds: 2 up, 2 in' (0)
root@addisu-mon1:~#
```

7. Sample ceph.conf configuration file used in the project.

This sample configuration is taken when 1 MON, 1 MDS, and 2 OSDs are used.

Sample /etc/ceph/ceph.conf

```
;
; Sample ceph ceph.conf file.
;
; This file defines cluster membership, the various locations
; that Ceph stores data, and any other runtime options.

; If a 'host' is defined for a daemon, the start/stop script will
; verify that it matches the hostname (or else ignore it).  If it is
; not defined, it is assumed that the daemon is intended to start on
; the current host (e.g., in a setup with a startup.conf on each
```

```
; node).

; global
[global]
        ; enable secure authentication
        ; auth supported = cephx

        ; allow ourselves to open a lot of files
        max open files = 131072

; monitors
;  You need at least one.  You need at least three if you want to
;  tolerate any node failures.  Always create an odd number.
[mon]
        mon data = /data/mon$id

        ; logging, for debugging monitor crashes, in order of
        ; their likelihood of being helpful :)
        debug ms = 1
        ;debug mon = 20
        ;debug paxos = 20
        ;debug auth = 20

[mon0]
        host = mon1
        mon addr = 10.0.0.8:6789

; mds
;  You need at least one.  Define two to get a standby.
[mds]

 ; where the mds keeps its secret encryption keys
        ; keyring = /data/keyring.$name

        ; mds logging to debug issues.
        debug ms = 1
        ;debug mds = 20

[mds.mds1]
        host = mds1

;[mds.mds2]
;       host = mds2

; osd
;  You need at least one.  Two if you want data to be replicated.
;  Define as many as you like.
[osd]
        ; This is where the btrfs volume will be mounted.
        osd data = /data/osd$id

        ; Ideally, make this a separate disk or partition.  A few
        ; hundred MB should be enough; more if you have fast or many
        ; disks.  You can use a file under the osd data dir if need be
        ; (e.g. /data/osd$id/journal), but it will be slower than a
        ; separate disk or partition.

        ; This is an example of a file-based journal.
        osd journal = /data/osd$id/journal
        osd journal size = 1000 ; journal size, in megabytes

        ; osd logging to debug osd issues, in order of likelihood of being
```

```
73          ; helpful
74          debug ms = 1
75          ;debug osd = 20
76          ;debug filestore = 20
77          ;debug journal = 20
78
79      [osd0]
80          host = osd1
81
82          ; if btrfs devs is not specified, you are responsible for
83          ; setting up the osd data dir.  if it is not btrfs, things
84          ; will behave up until you try to recover from a crash (which
85          ; usually fine for basic testing).
86          btrfs devs = /dev/sdb5
87
88      [osd1]
89          host = osd2
90          btrfs devs = /dev/sdb5
91
```

Appendix D

Complete code of three Perl scripts used

Complete code of three Perl scripts used:

1. The Perl script used to make the 50 VMs and the server hosting them ready for benchmarking:

The 1st Perl script used (main portion)

```
system("mkdir /mnt/ceph");
system("mount -t ceph mon1:/ /mnt/ceph");

for($i=1; $i<=50; $i++)
{
   system("mkdir /mnt/ceph/CLIENT_0$i") if ($i<=9);
   system("mkdir /mnt/ceph/CLIENT_$i") if ($i>=10);
}

for($i=1; $i<=50; $i++)
{
system("ssh root\@client$i \'apt-get update; apt-get install iozone3 ceph\'");
system("ssh root\@client$i \'mkdir /mnt/ceph\'");
system("ssh root\@client$i \'mount -t ceph mon1:/ /mnt/ceph\'");
}
```

2. The 2nd Perl script used to save cron jobs in all VMs to be used for benchmarking concurrently

The 2nd Perl script used

```
#!/usr/bin/perl

############################################################
# This Perl script saves cron jobs in all /etc/crontab    #
# files of all client VMs to be used benchmarking ceph    #
# DFS concurrently. It also writes and saves SHELL        #
# scripts in all client VMs to be run by cron             #
############################################################

use Getopt::Std;
```

```
use strict "vars";

my $VERBOSE = 0;
my $DEBUG = 0;

```

Flags and Arguments

```
my $opt_string = 'vdhn:i:m:H:D:M:w:';
getopts( "$opt_string", \my %opt ) or usage() and exit 1;

if ( $opt{'h'} ) {
    usage();
    exit 0;
}

my $VERBOSE = 1 if $opt{'v'};
my $DEBUG = 1 if $opt{'d'};

my $vms = $opt{'n'} if $opt{'n'};
my $itrn = $opt{'i'} if $opt{'i'};
my $min = $opt{'m'} if $opt{'m'};
my $hr = $opt{'H'} if $opt{'H'};
my $date = $opt{'D'} if $opt{'D'};
my $mon = $opt{'M'} if $opt{'M'};
my $wk = $opt{'w'} if $opt{'w'};

```

Saving cron jobs

```
my $v;
for($v=1; $v<=9;$v++)
{
system("touch /root/SHELL/cephIO_for_$vms\VMs_onVM0$v.sh");
system("echo \'#!/bin/bash\' >
 /root/SHELL/cephIO_for_$vms\VMs_onVM0$v.sh");
system("echo \'iozone -i 1 -i 0 -r 256 -s 1g >
cephIO_for_$vms\VMs_onVM0$v.txt$itrn\'>>
/root/SHELL/cephIO_for_$vms\VMs_onVM0$v.sh");

system("scp /root/SHELL/cephIO_for_$vms\VMs_onVM0$v.sh
 root\@client$v:/root/cephIO_for_$vms\VMs_onVM0$v.sh");

system("ssh root\@client$v \"chmod 755
 /root/cephIO_for_$vms\VMs_onVM0$v.sh\"");

system("ssh root\@client$v \"echo \'$min $hr $date $mon
 $wk root (cd /mnt/ceph/CLIENT_0$v &&
  /root/cephIO_for_$vms\VMs_onVM0$v.sh)\'>>/etc/crontab\"");
}

for($v=10; $v<=$vms;$v++)
{
system("touch /root/SHELL/cephIO_for_$vms\VMs_onVM$v.sh");
system("echo \'#!/bin/bash\'> /root/SHELL/cephIO_for_$vms\VMs_onVM$v.sh");
system("echo \'iozone -i 1 -i 0 -r 256 -s 1g >
cephIO_for_$vms\VMs_onVM$v.txt$itrn\'>>
/root/SHELL/cephIO_for_$vms\VMs_onVM$v.sh");
system("scp /root/SHELL/cephIO_for_$vms\VMs_onVM$v.sh
root\@client$v:/root/cephIO_for_$vms\VMs_onVM$v.sh");
system("ssh root\@client$v \"chmod 755 \
```

```
32
33  /root/cephIO_for_$vms\VMs_onVM$v.sh\"");
34      system("ssh root\@client$v \"echo \'$min $hr $date $mon
35      $wk root (cd /mnt/ceph/CLIENT_$v &&
36       /root/cephIO_for_$vms\VMs_onVM$v.sh)\'>>/etc/crontab\"");
37  }
```

Helps script usage

```
1   sub usage {
2       print "Usage:\n";
3       print "-h Usage\n";
4       print "-v Verbose\n";
5       print "-d Debug\n";
6       print "-n Number of VMs\n";
7       print "-i iteration\n";
8       print "-m cron minute\n";
9       print "-H cron hour\n";
10      print "-D cron date\n";
11      print "-M cron month\n";
12      print "-w cron week\n";
13  }
14
15  sub verbose {
16      print $_[0] if ($VERBOSE or $DEBUG);
17  }
18
19  sub debug {
20      print $_[0] if ($DEBUG);
21  }
22
```

3. The 3rd Perl script used to collect data.

The 3rd Perl script used

```
1   #! /usr/bin/perl
2
3   ############################
4   # This Perl script collects iozone outputs   #
5   ############################
6
7   use Getopt::Std;
8   use strict "vars";
9
10  my $VERBOSE = 0;
11  my $DEBUG = 0;
12
```

Flags and Arguments

```
1   my $opt_string = 'vdhn:m:H:D:M:w:'; # h-help, v-verbose, d-debug
2   getopts( "$opt_string", \my %opt ) or usage() and exit 1; # exit 1 is for error,
3
4   # print help message if -h is invoked
5
6   if ( $opt{'h'} ) {
7       usage();
8       exit 0; # proper exit. exit without number takes 0 as a default.
```

```
}

my $VERBOSE = 1 if $opt{'v'};
my $DEBUG = 1 if $opt{'d'};

my $vms = $opt{'n'} if $opt{'n'};
my $min = $opt{'m'} if $opt{'m'};
my $hr = $opt{'H'} if $opt{'H'};
my $date = $opt{'D'} if $opt{'D'};
my $mon = $opt{'M'} if $opt{'M'};
my $wk = $opt{'w'} if $opt{'w'};
```

Saving cron jobs

```
my $i;
for($i=1; $i<=9;$i++)
{
system("touch /root/SHELL/cephIO_for_$vms\VMs_onVM0 \
$i.sh");
system("echo \'#!/bin/bash\' > \
 /root/SHELL/cephIO_for_$vms\VMs_onVM0 \
$i.sh");
system("echo \'iozone -i 1 -i 0 -r 256 -s 1g > \
 cephIO_for_$vms\VMs_onVM0$i.txt\' \
 >>/root/SHELL/cephIO_for_$vms\VMs_onVM0$i.sh");
system("scp /root/SHELL/cephIO_for_$vms\VMs_onVM0$i.sh
 root\@client$i:/root
/cephIO_for_$vms\VMs_onVM0$i.sh");
system("ssh root\@client$i \"chmod 755
    /root/cephIO_for_$vms\VMs_onVM0$i.sh\"");
system("ssh root\@client$i \"echo \'$min $hr $date $mon $wk \
root (cd /mnt/ceph/CLIENT_0$i &&
 /root/cephIO_for_$vms\VMs_onVM0$i.sh)'>> /etc/crontab\"");
}

for($i=10; $i<=$vms;$i++)
{
system("touch /root/SHELL/cephIO_for_$vms\VMs_onVM$i.sh");
system("echo \'#!/bin/bash\' >
 /root/SHELL/cephIO_for_$vms\VMs_onVM$i.sh");
system("echo \'iozone -i 1 -i 0 -r 256 -s 1g >
 cephIO_for_$vms\VMs_onVM$i.txt\'>>\
/root/SHELL/cephIO_for_$vms\VMs_onVM$i.sh");
system("scp /root/SHELL/cephIO_for_$vms\VMs_onVM$i.sh
root\@client$i:/root/cephIO_for_$vms\VMs_onVM$i.sh");
system("ssh root\@client$i \"chmod 755
 /root/cephIO_for_$vms\VMs_onVM$i.sh\"");
system("ssh root\@client$i \"echo \'$min $hr $date $mon $wk
root (cd /mnt/ceph/CLIENT_$i &&
/root/cephIO_for_$vms\VMs_onVM$i.sh)\'>> \
/etc/crontab\"");
}
```

Help script usage

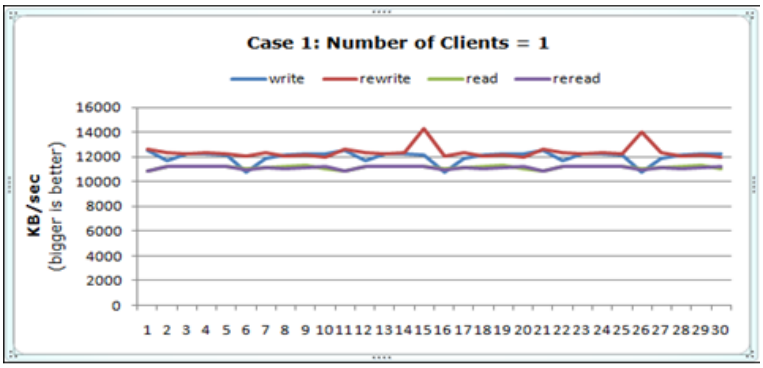
```
sub usage {
    print "Usage:\n";
    print "-h Usage\n";
    print "-v Verbose\n";
    print "-d Debug\n";
    print "-n Number of VMs\n";
```

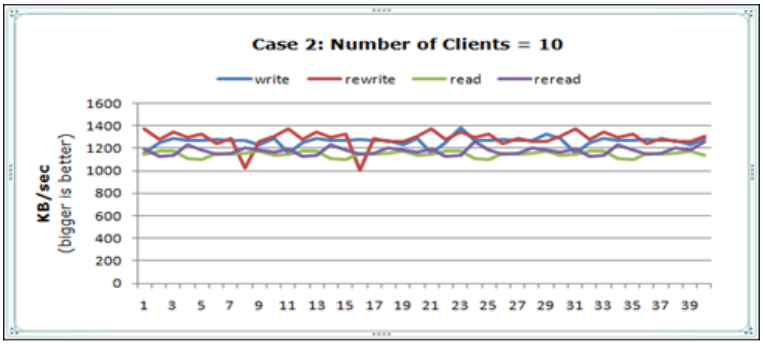
```
7       print "-m cron minute\n";
8       print "-H cron hour\n";
9       print "-D cron date\n";
10      print "-M cron month\n";
11      print "-w cron week\n";
12  }
13
14  sub verbose {
15      print $_[0] if ($VERBOSE or $DEBUG);
16  }
17
18  sub debug {
19      print $_[0] if ($DEBUG);
20  }
```

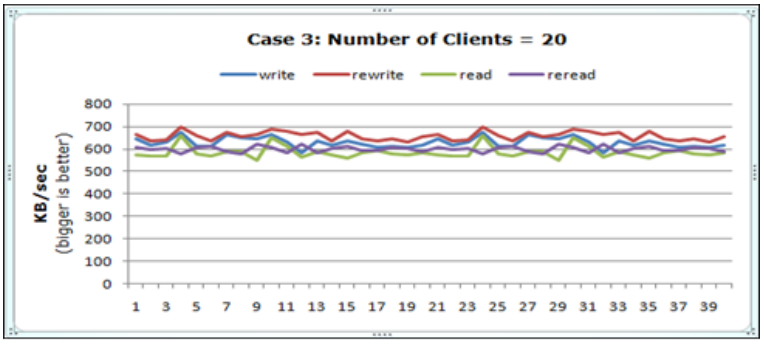
Appendix E

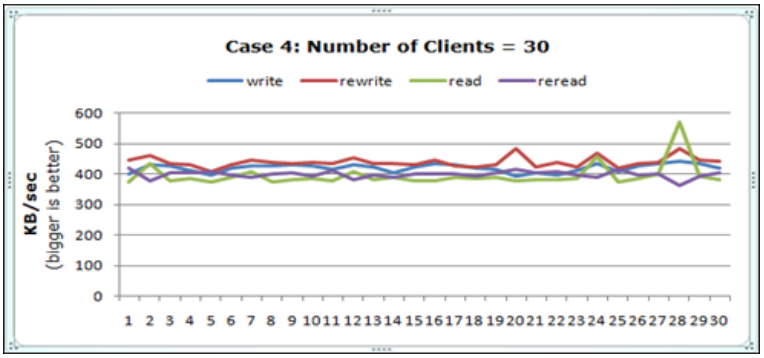
Supplementary graphs of benchmarking results

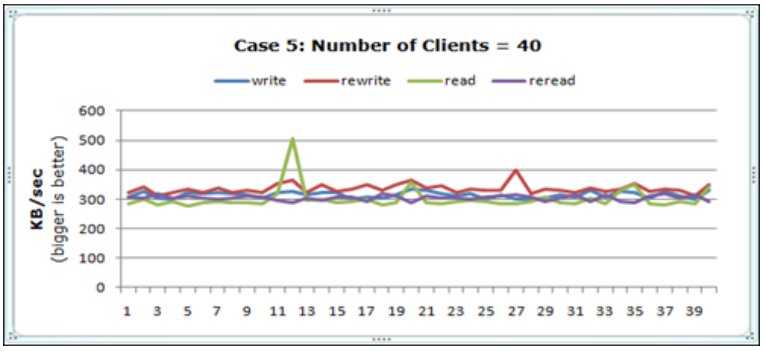
Supplementary graphs of benchmarking results:

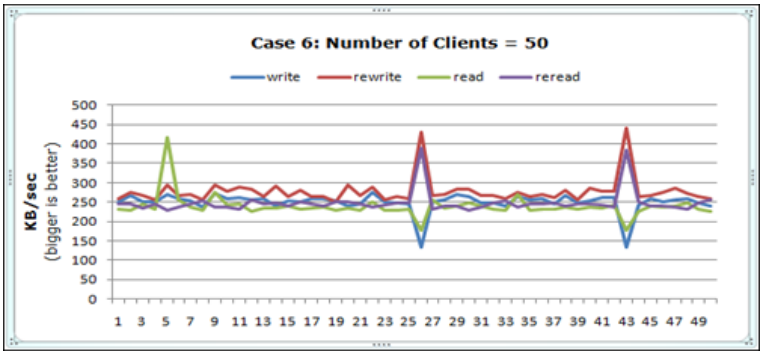


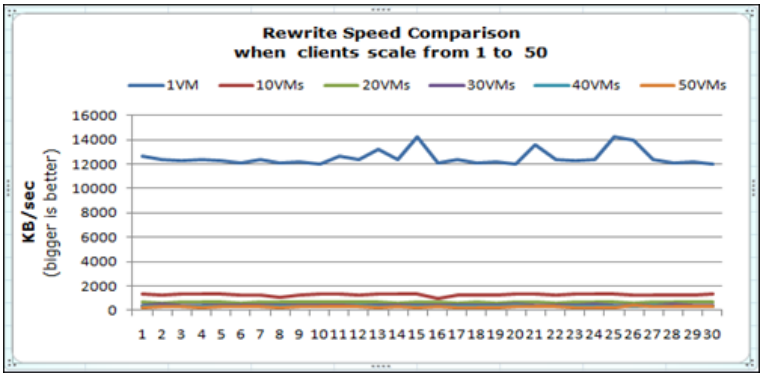


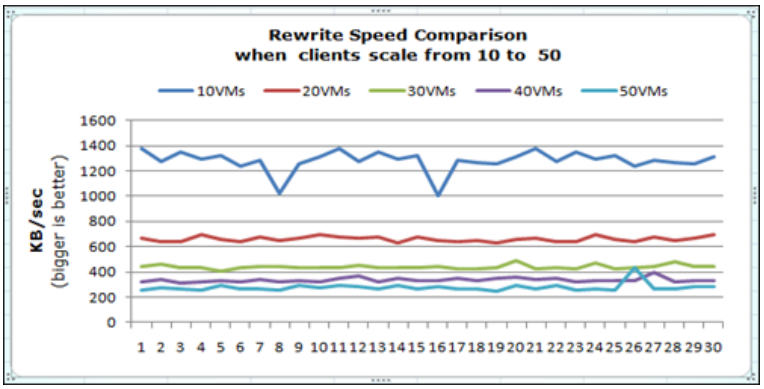

Case 3: Number of Clients = 20
write
rewrite
read
reread
KB/sec
(bigger is better)
800
700
600
500
400
300
200
100
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39

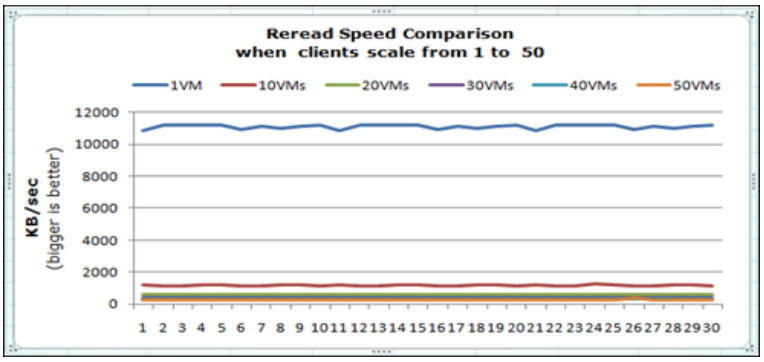

Case 4: Number of Clients = 30
write
rewrite
read
reread
KB/sec
(bigger is better)
600
500
400
300
200
100
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

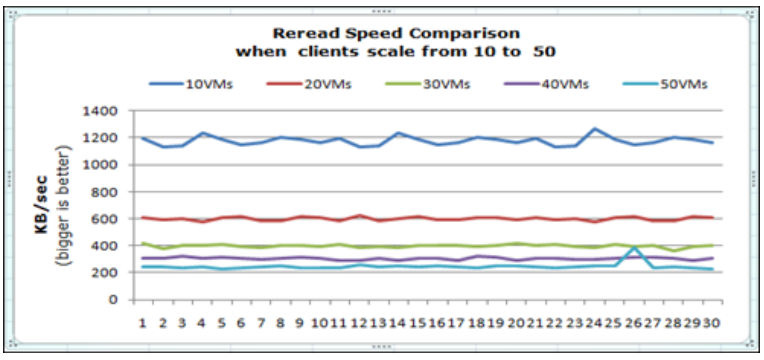

Case 5: Number of Clients = 40
write
rewrite
read
reread
KB/sec
(bigger is better)
600
500
400
300
200
100
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39

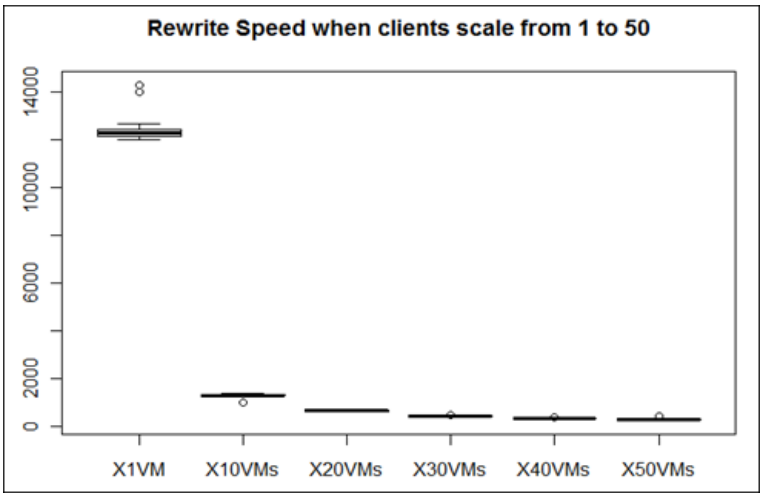

Case 6: Number of Clients = 50
write
rewrite
read
reread
KB/sec
(bigger is better)
500
450
400
350
300
250
200
150
100
50
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49

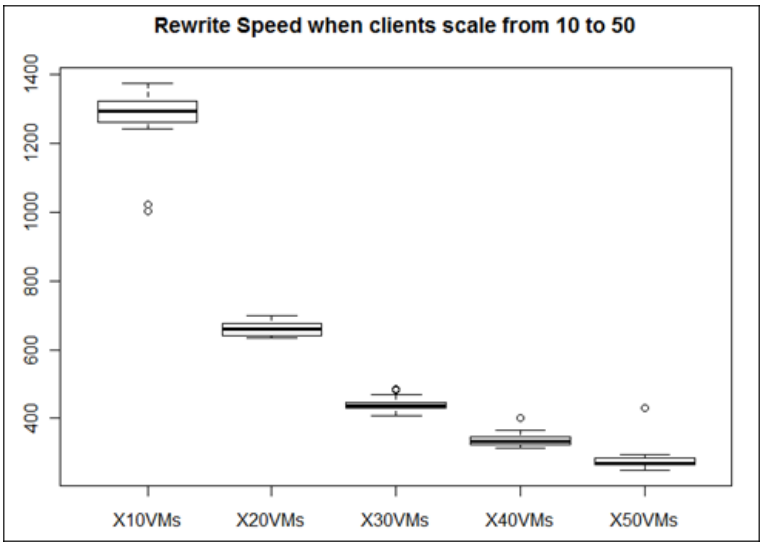

Rewrite Speed Comparison
when clients scale from 1 to 50
1VM
10VMs
20VMs
30VMs
40VMs
50VMs
KB/sec
(bigger is better)
16000
14000
12000
10000
8000
6000
4000
2000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

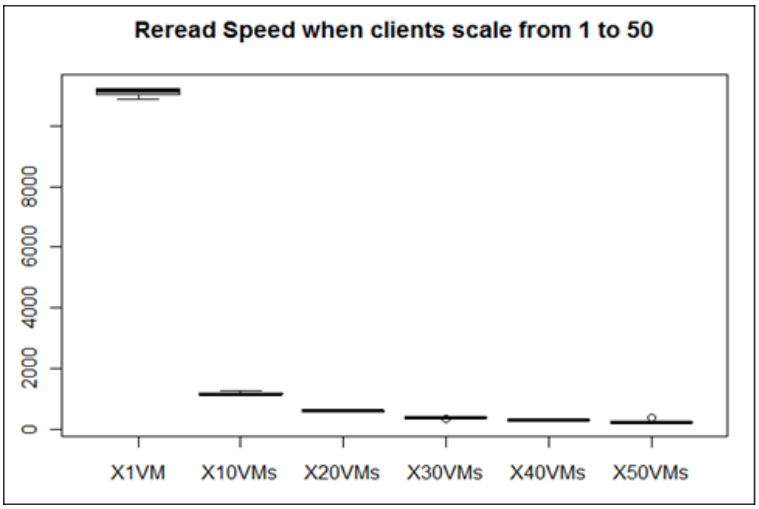

Rewrite Speed Comparison
when clients scale from 10 to 50
10VMs
20VMs
30VMs
40VMs
50VMs
KB/sec
(bigger is better)
1600
1400
1200
1000
800
600
400
200
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

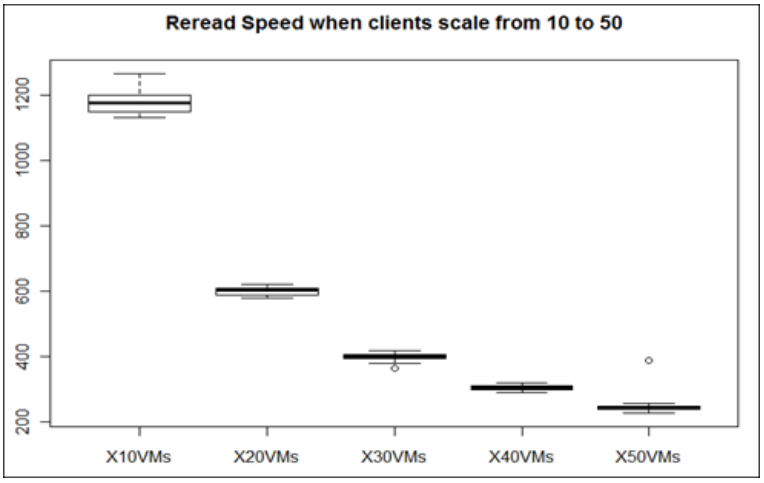

Reread Speed Comparison
when clients scale from 1 to 50
1VM
10VMs
20VMs
30VMs
40VMs
50VMs
KB/sec
(bigger is better)
12000
10000
8000
6000
4000
2000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30


Reread Speed Comparison
when clients scale from 10 to 50
10VMs
20VMs
30VMs
40VMs
50VMs
KB/sec
(bigger is better)
1400
1200
1000
800
600
400
200
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30


Rewrite Speed when clients scale from 1 to 50
0
2000
6000
10000
14000
X1VM
X10VMs
X20VMs
X30VMs
X40VMs
X50VMs


Rewrite Speed when clients scale from 10 to 50
400
600
800
1000
1200
1400
X10VMs
X20VMs
X30VMs
X40VMs
X50VMs


Reread Speed when clients scale from 1 to 50
0
2000
4000
6000
8000
X1VM
X10VMs
X20VMs
X30VMs
X40VMs
X50VMs


Reread Speed when clients scale from 10 to 50
200
400
600
800
1000
1200
X10VMs
X20VMs
X30VMs
X40VMs
X50VMs

Appendix F

Acronyms

Acronyms:

* **DFS:** Distributed File System
* **OSD:** Object Storage Device
* **MDS:** MetaData server
* **MON:** Monitor
* **RADOS:** Reliable Autonomic Distributed Object Store
* **PG:** Placement Group
* **POSIX:** Portable Operating System Interface for UNIX
* **CC:** Cloud Computing
* **Btrfs:** B-tree or Butter file system
* **NFS:** Network File System
* **FFS:** Fast File System
* **NAS:** Network Attached Storage
* **SAN:** Storage Area Network
* **pNFS:** Parallel Network File System
* **GFS:** Google File System
* **GPFS:** General Parallel File System
* **CRUSH:** Controlled Replication Under Scalable Hashing
* **cauthtool:** **C**eph **auth**entication **tool**

* **EBOFS:** Extent and B-tree based Object File System
* **HPC:** High Performance Computing
* **UFS:** UNIX File System
* **EFS:** Encrypting File System
* **SMB:** Server Message Block
* **CIFS:** Common Internet File System

Bibliography

[1] http://ceph.newdream.net. 2011.

[2] https://btrfs.wiki.kernel.org. 2011.

[3] http://www.gnuplot.info. 2011.

[4] http://www.ibm.com/developerworks/linux/library/l-ceph. 2011.

[5] http://www.iozone.org. 2011.

[6] http://www.pnfs.com. 2011.

[7] www.gluster.org. 2011.

[8] Pete Wyckoff Ananth Devulapalli. File creation strategies in a distributed metadata file system.

[9] Pete Wyckoff Nawab Ali Ananth Devulapalli, Dennis Dalessandro. Attribute storage design for object-based storage devices. 2007.

[10] Rogier Spoor Paul Dekkers Christiaan den Besten Arjan Peddemors, Christiaan Kuun. Survey of technologies for wide area distributed storage. 2010.

[11] Sage A.Weil. Ceph: Reliable, scalable, and high-performance distributed storage. 2007.

[12] AMANDEEP KHURANA ALEX J. NELSON SCOT T A. BRANDT CARLOS MALTZAHN, ESTEBAN MOL INAESTOLANO and SAGE WEIL. Ceph as a scalable alternative to the hadoop distributed file system. 2010.

[13] Unisys Corporation. Linux file systems comparative performance. 2005.

[14] LING-FANG ZENG QUN LIU DAN FENG, LINGJUN QIN. A scalable object-based intelligent storage device. 2004.

[15] Peter Honeyman Dean Hildebrand. Exporting storage systems in a scalable manner with pnfs. 2006.

[16] Galen Shipman Oleg Drokin Tom Wang Isaac Huang Feiyi Wang, Sarp Oral. Understanding lustre filesystem internals. 2009.

[17] Machtelt Garrels. Introduction to linux: A hands on guide. 2008.

[18] Christopher R. Hertel. Implementing cifs the common inernet filesystem. 2003.

[19] Qinghua Gao Dawei Huang Jianhua Che, Qinming He. Performance measuring and comparing of virtual machine monitors*. 2008.

[20] Paul Krzyzanowski. Lectures on distributed systems distributed file systems design. 2002.

[21] Bjørn Erik Lømo. File system supporting arbitrarily sized allocation. Master's thesis, University Of Oslo, 2008.

[22] Christian Lunden. Evaluation of different san technologies for virtual machine hosting. Master's thesis, University Of Oslo, 2009.

[23] Sage A. Weil Andrew W. Leung Scott A. Brandt Carlos Maltzahn. Rados: A scalable, reliable storage service for petabyte-scale storage clusters.

[24] Sage A. Weil Scott A. Brandt Ethan L. Miller Darrell D. E. Long Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. 2006.

[25] Casey Marshall. Object storage on berkeley db1. 2007.

[26] Sun microsystems. File system performance: The solaris os, ufs, linux ext3,and reiserfs. 2004.

[27] Sun microsystems. High-performance storage architecture and scalable cluster file system. 2007.

[28] Christian Mikalsen. Moving into the cloud. Master's thesis, University Of Oslo, 2009.

[29] Garth Gibson Milo Polte, Jiri Simsa. Comparing performance of solid state devices and mechanical disks. 2008.

[30] Dennis Dalessandro Pete Wyckoff P. Sadayappan Nawab Ali, Ananth Devulapalli. An osd-based approach to managing directory operations in parallel file systems. 2008.

[31] John Hawkes Ray Bryant, Ruth Forester. Filesystem performance and scalability in linux 2.4.17. 2002.

[32] Thomas M. Ruwart. File system benchmarks, then, now, and tomorrow. 2003.

[33] Howard Gobioff Sanjay Ghemawat and Shun-Tak Leung. The google file system. 2003.

[34] Brad Schonhorst. Evolution of the unix file system. 2006.

[35] Thijs Stuurman. pnfs: Parallel network file system. 2008.

[36] Viet-Trung TRAN. Using object-based storage to build distributed file systems: a survey. 2009.

[37] Naushad UzZaman. Survey on google file system. 2007.

[38] Andrea C. Arpaci-Dusseau Vijayan Prabhakaran and Remzi H. Arpaci-Dusseau. Analysis and evolution of journaling file systems. 2005.

[39] John Enok Vollestad. A high performance cluster file system using sci. Master's thesis, University Of Oslo, 2002.

[40] Jeffrey S. Vetter Weikuan Yu, Oleg Drokin. Design, implementation, and evaluation of transparent pnfs on lustre. 2009.

[41] Sage A. Weil. Leveraging intra-object locality with ebofs.

[42] Tiancheng Liu Jie Qiu Fengchun Wang Xinhui Li, Ying Li. The method and tool of cost analysis for cloud computing. 2009.

[43] Dan Feng Yinliang Yue, Fang Wang*. irbo: Intelligent role-based object for object-based storage device. 2007.